

A Source Book from The Open Group

**The Single UNIX[®] Specification:
Authorized Guide to Version 4**

The Open Group

Copyright © March 2010, The Open Group

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the copyright owners.

A Source Book from The Open Group

The Single UNIX[®] Specification: Authorized Guide to Version 4

ISBN: 1-931624-84-4

Document Number: G101

Published by The Open Group, March 2010.

Any comments relating to the material contained in this document may be submitted by email to:

OGspecs@opengroup.org

Contents

Chapter 1	The Single UNIX Environment.....	1
1.1	Introduction.....	1
1.2	Interface Counts	2
1.3	Formal Standards Alignment.....	2
1.4	Portability Codes	2
1.5	Option Groups	6
1.6	Common Directories and Devices	7
1.7	Environment Variables.....	7
1.8	YACC Grammars as Specifications	8
1.9	Regular Expressions	9
1.10	File Access	9
1.11	Programming Environment.....	9
1.11.1	C-Language Support	9
1.11.2	Feature Test Macros and Name Space Issues.....	10
1.11.3	Error Numbers	10
1.11.4	Signal Concepts	10
1.11.5	Standard I/O Streams.....	11
1.11.6	STREAMS	11
1.11.7	XSI Interprocess Communication	11
1.11.8	Realtime	11
1.11.9	Threads	12
1.11.10	Sockets	12
1.11.11	General Terminal Interface	12
1.11.12	How to Read an XSH Reference Page.....	12
1.12	Commands and Utilities Environment	13
1.12.1	Shell Command Language	13
1.12.2	Symbolic Links.....	13
1.12.3	File Format Notation	13
1.12.4	How to Read an XCU Reference Page	14
1.13	Terminal Interfaces Environment	14
1.14	Internationalization	15
Chapter 2	The Single UNIX Specification, Version 4.....	17
2.1	Base Definitions (XBD).....	17
2.2	System Interfaces (XSH)	18
2.3	Shell and Utilities (XCU).....	18
2.4	Rationale (XRAT)	19
2.5	X/Open Curses (XCURSES)	19

Chapter	3	System Interfaces	21
	3.1	Base Documents	21
	3.2	Overview of Changes	21
	3.3	System Interfaces by Category	25
	3.4	XSH Option Groups	32
	3.5	Options Policy	34
Chapter	4	Shell and Utilities	37
	4.1	Options in XCU	37
	4.2	Functional Overview	37
	4.3	Base Documents	39
	4.4	Overview of Changes	39
	4.5	Exclusion of Utilities.....	40
Chapter	5	Headers	45
	5.1	Header and Name Space Rules.....	45
	5.1.1	ISO C Headers	45
	5.1.2	POSIX.1-2008 Base Headers.....	46
	5.1.3	XSI Headers	47
	5.2	Names Safe to Use	48
	5.3	Base Documents	48
	5.4	Overview of Changes	48
Chapter	6	Terminal Interfaces	51
	6.1	Functional Overview	51
	6.1.1	Curses Interfaces	51
	6.1.2	Curses Utilities.....	53
	6.2	Overview of Changes	53
Chapter	7	System Interface Table	55
	7.1	Introduction.....	55
	7.2	System Interface Table	56
Chapter	8	Utility Interface Table	81
	8.1	Introduction.....	81
	8.2	Utility Interface Table	82
Chapter	9	Header Interface Table	87
	9.1	Introduction.....	87
	9.2	Header Interface Table	88
Chapter	10	XCURSES Interface Table	91
	10.1	Introduction.....	91
	10.2	XCURSES Interface Table.....	91
Chapter	11	System Interfaces Migration	101
	11.1	Introduction.....	101
	11.2	System Interfaces.....	101
Chapter	12	Utilities Migration	299
	12.1	Introduction.....	299
	12.2	Utilities	299

Chapter 13	Headers Migration	341
13.1	Introduction	341
13.2	Headers	341
Chapter 14	ISO C Migration	363
14.1	Introduction	363
14.2	Language Changes	363
14.2.1	New Keywords	363
14.2.2	New Types	364
14.2.3	Type Qualifiers	364
14.2.4	Boolean	365
14.2.5	Universal Character Names.....	365
14.2.6	inline	365
14.2.7	Predefined Identifiers.....	366
14.2.8	Compound Literals	366
14.2.9	Designated Initializers	366
14.3	Decimal Integer Constants	367
14.3.1	String Literals.....	367
14.4	Implicit Declarations	367
14.4.1	sizeof	367
14.4.2	Multiplicative Operators	367
14.4.3	Enumeration Specifiers	368
14.5	Variable Length Array	368
14.5.1	Array Declarations	368
14.5.2	Array Type Compatibility	368
14.5.3	Incomplete Array Structure Members.....	368
14.5.4	Blocks	369
14.5.5	The for Statement.....	369
14.5.6	errno	369
14.6	Comments	370
14.6.1	Hexadecimal Floating-Point Constants.....	370
14.6.2	Predefined Macros	370
14.6.3	Source File Inclusion	370
14.6.4	Translation-Time Arithmetic.....	370
14.6.5	Minimum Maximum Line Length.....	371
14.6.6	Case-Sensitive Identifiers.....	371
14.6.7	line Directive	371
14.6.8	Empty Argument Macros	371
14.6.9	Pragmas	371
14.6.10	Translation Limits.....	371
14.6.11	Token Pasting	372
14.6.12	Variadic Macros	372
14.6.13	va_copy()	372
14.7	Headers	373
14.8	Integer Types	373
14.8.1	Exact-Width Integer Types.....	373
14.8.2	Minimum-Width Integer Types	374
14.8.3	Fastest Minimum-Width Integer Types	374
14.8.4	Integer Types Capable of Holding Object Pointers	374
14.8.5	Greatest-Width Integer Types.....	374
14.8.6	Limits of Specified-Width Integer Types	376
14.8.7	Macros.....	376

14.9	Complex Numbers	376
14.9.1	Trigonometric Functions	377
14.9.2	Hyperbolic Functions	378
14.9.3	Exponential and Logarithmic Functions.....	378
14.9.4	Power and Absolute-Value Functions.....	379
14.9.5	Manipulation Functions.....	379
14.10	Other Mathematical Changes.....	380
14.10.1	Classification Macros.....	380
14.10.2	Trigonometric Functions	381
14.10.3	Hyperbolic Functions	381
14.10.4	Exponential and Logarithmic Functions.....	382
14.10.5	Nearest Integer Functions	384
14.10.6	Remainder Functions	385
14.10.7	Manipulation Functions.....	385
14.10.8	Comparison Macros	386
14.11	Floating-Point Environment Support.....	387
14.11.1	Exceptions	387
14.11.2	Rounding	388
14.11.3	Environment	388
14.12	Type-Generic Math	389
14.12.1	Unsuffix Functions With a C-Prefixed Counterpart	389
14.12.2	Unsuffix Functions Without a C-Prefixed Counterpart	389
14.13	Other Library Changes	390
14.13.1	Wide-String Numeric Conversion Functions.....	393
14.14	Annexes.....	394
	Index	395

About This Document

This document is designed for users, developers, and implementors of open systems. It provides complete information on what's new in the Single UNIX Specification, Version 4, including POSIX.1-2008, with comprehensive reference material on every aspect.

This document is structured as follows:

- **Chapter 1** describes how the Single UNIX Specification environment is organized and where to look for more detailed information in the Single UNIX Specification.
- **Chapter 2** gives an overview of the documents that comprise the different parts of the Single UNIX Specification, Version 4, and how they are organized. This includes an overview of the contents of each of the documents and their relation to each other.
- **Chapter 3** covers the C language programming interfaces that are the system interfaces, describing the base documents, an overview of the changes in this version, and lists of the new functions. It also describes the options and groups the interfaces into categories to give the reader an understanding of what facilities are available.
- **Chapter 4** covers the shell and utilities, describing the base documents, an overview of the changes in this version, and lists of the new features. It also contains rationale of why certain utilities are excluded, and also a description of the options.
- **Chapter 5** covers the C language headers and name space rules. It describes the base documents, and gives an overview of the changes in this version.
- **Chapter 6** describes the contents of X/Open Curses, Issue 7.
- **Chapter 7** gives a quick reference table of all the system interfaces defined in the Base Specifications, Issue 7, complete with an indication of their status for XSI-conforming and POSIX-conforming systems, and their availability in UNIX 03, UNIX 98, UNIX 95, POSIX.1-2001, POSIX.1-1996, POSIX.2-1992, C99, and C89.
- **Chapter 8** gives a quick reference table of all the utility interfaces defined in the Base Specifications, Issue 7, complete with an indication of their status for XSI-conforming and POSIX-conforming systems, and their availability in UNIX 03, UNIX 98, UNIX 95, POSIX.1-2001, and POSIX.2-1992.
- **Chapter 9** gives a quick reference table of the headers defined in the Base Specifications, Issue 7, complete with an indication of their status for XSI-conforming and POSIX-conforming systems, and their availability in UNIX 03, UNIX 98, UNIX 95, POSIX.1-2001, POSIX.1-1996, POSIX.2-1992, C99, and C89.
- **Chapter 10** gives a quick reference table of the XCURSES interfaces, complete with an indication of their availability in other specifications.
- **Chapter 11** provides migration information on the changes made to system interfaces in Issue 7 that might affect an application program.

- **Chapter 12** provides migration information on the changes made to the utilities in Issue 7 that might affect an application program.
- **Chapter 13** provides migration information on the changes made to the headers in Issue 7 that might affect an application program.
- **Chapter 14** provides migration information on the changes made to the ISO C Language for the ISO/IEC 9899: 1999 standard.

About The Open Group

The Open Group

The Open Group is a vendor-neutral and technology-neutral consortium, whose vision of Boundaryless Information Flow™ will enable access to integrated information within and between enterprises based on open standards and global interoperability. The Open Group works with customers, suppliers, consortia, and other standards bodies. Its role is to capture, understand, and address current and emerging requirements, establish policies, and share best practices; to facilitate interoperability, develop consensus, and evolve and integrate specifications and Open Source technologies; to offer a comprehensive set of services to enhance the operational efficiency of consortia; and to operate the industry's premier certification service, including TOGAF™ and UNIX® certification.

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

The Open Group has over 20 years' experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification.

More information is available at www.opengroup.org/certification.

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical and Product Standards and Guides, but which also includes white papers, technical studies, branding and testing documentation, and business titles. Full details and a catalog are available at www.opengroup.org/bookstore.

As with all *live* documents, Technical Standards and Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards-compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.
- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published at www.opengroup.org/corrigenda.

About the Austin Group

The Austin Group is a joint working Group of members of the IEEE Portable Applications Standards Committee, members of The Open Group, and members of ISO/IEC Joint Technical Committee 1 (JTC1). The Austin Group continues as the maintenance body for the specification that is ISO/IEC 9945, IEEE Std 1003.1, and The Open Group Base Specifications.

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

Trademarks

Boundaryless Information Flow™ and TOGAF™ are trademarks and Motif®, Making Standards Work®, OSF/1®, The Open Group®, UNIX®, and the “X” device are registered trademarks of The Open Group in the United States and other countries.

POSIX® is a registered trademark of the Institute of Electrical and Electronic Engineers, Inc.

The Open Group acknowledges that there may be other company names and products that might be covered by trademark protection and advises the reader to verify them independently.

Acknowledgements

The Open Group gratefully acknowledges:

- The Austin Group for developing the Base Specifications, Issue 7
- The Base Working Group for developing XCURSES, Issue 7
- Finnbar P. Murphy for writing [Chapter 14](#), the ISO C Language migration chapter

Referenced Documents

The following documents are referenced in this guide:

ANSI C

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

HP-UX Manual

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

IEC 60559: 1989

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

ISO/IEC 10646-1: 2000

ISO/IEC 10646-1: 2000, Information Technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.

ISO/IEC 10967-1: 1994

ISO/IEC 10967-1: 1994, Information Technology — Language-Independent Arithmetic — Part 1: Integer and Floating-Point Arithmetic.

ISO C (1990)

ISO/IEC 9899: 1990, Programming Languages — C, including Amendment 1: 1995 (E), C Integrity (Multibyte Support Extensions (MSE) for ISO C).

ISO C (1999)

ISO/IEC 9899: 1999, Programming Languages — C, including ISO/IEC 9899: 1999/Cor.1: 2001(E), ISO/IEC 9899: 1999/Cor.2: 2004(E), and ISO/IEC 9899: 1999/Cor.3(E)

ISO POSIX-1: 1996

ISO/IEC 9945-1: 1996, Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language] (identical to ANSI/IEEE Std 1003.1-1996). Incorporating ANSI/IEEE Stds 1003.1-1990, 1003.1b-1993, 1003.1c-1995, and 1003.1i-1995.

ISO POSIX-2: 1993

ISO/IEC 9945-2: 1993, Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities (identical to ANSI/IEEE Std 1003.2: 1992, as amended by ANSI/IEEE Std 1003.2a: 1992).

Issue 1

X/Open Portability Guide, July 1985 (ISBN: 0-444-87839-4).

Issue 2

X/Open Portability Guide, January 1987:

- Volume 1: XVS Commands and Utilities (ISBN: 0-444-70174-5)

- Volume 2: XVS System Calls and Libraries (ISBN: 0-444-70175-3)

Issue 3

X/Open Specification, 1988, 1989, February 1992:

- Commands and Utilities, Issue 3 (ISBN: 1-872630-36-7, C211); this specification was formerly X/Open Portability Guide, Issue 3, Volume 1, January 1989, XSI Commands and Utilities (ISBN: 0-13-685835-X, XO/XPG/89/002)
- System Interfaces and Headers, Issue 3 (ISBN: 1-872630-37-5, C212); this specification was formerly X/Open Portability Guide, Issue 3, Volume 2, January 1989, XSI System Interface and Headers (ISBN: 0-13-685843-0, XO/XPG/89/003)
- Curses Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapters 9 to 14 inclusive; this specification was formerly X/Open Portability Guide, Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)
- Headers Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapter 19, Cpio and Tar Headers; this specification was formerly X/Open Portability Guide Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)

Issue 4

CAE Specification, July 1992, published by The Open Group:

- System Interface Definitions (XBD), Issue 4 (ISBN: 1-872630-46-4, C204)
- Commands and Utilities (XCU), Issue 4 (ISBN: 1-872630-48-0, C203)
- System Interfaces and Headers (XSH), Issue 4 (ISBN: 1-872630-47-2, C202)

Issue 4, Version 2

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

- System Interface Definitions (XBD), Issue 4, Version 2 (ISBN: 1-85912-036-9, C434)
- Commands and Utilities (XCU), Issue 4, Version 2 (ISBN: 1-85912-034-2, C436)
- System Interfaces and Headers (XSH), Issue 4, Version 2 (ISBN: 1-85912-037-7, C435)

Issue 5

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

- System Interface Definitions (XBD), Issue 5 (ISBN: 1-85912-186-1, C605)
- Commands and Utilities (XCU), Issue 5 (ISBN: 1-85912-191-8, C604)
- System Interfaces and Headers (XSH), Issue 5 (ISBN: 1-85912-181-0, C606)

Issue 6

Technical Standard, December 2001, published by The Open Group:

- Base Definitions (XBD), Issue 6, 2004 Edition (ISBN: 1-931624-43-7, C046)
- System Interfaces (XSH), Issue 6, 2004 Edition (ISBN: 1-931624-44-5, C047)
- Shell and Utilities (XCU), Issue 6, 2004 Edition (ISBN: 1-931624-45-3, C048)
- Rationale (Informative), Issue 6, 2004 Edition (ISBN: 1-931624-46-1, C049)

(See also POSIX.1: 2001.)

Referenced Documents

Issue 7

Technical Standard, December 2008, Base Specifications, Issue 7 (ISBN: 1-931624-79-8, C082), published by The Open Group.

(See also POSIX.1:2008.)

MSE Working Draft

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

POSIX.1: 1988

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

POSIX.1: 1990

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

POSIX.1: 1996

IEEE Std 1003.1-1996, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language].

POSIX.1: 2001

IEEE Std 1003.1-2001, IEEE Standard for Information Technology — Portable Operating System.

(See also Issue 6.)

POSIX.1: 2008

IEEE Std 1003.1-2008, IEEE Standard for Information Technology — Portable Operating System.

(See also Issue 7.)

POSIX.1a

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

POSIX.1d: 1999

IEEE Std 1003.1d-1999, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 4: Additional Realtime Extensions [C Language].

POSIX.1g: 2000

IEEE Std 1003.1g-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 6: Protocol-Independent Interfaces (PII).

POSIX.1j: 2000

IEEE Std 1003.1j-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 5: Advanced Realtime Extensions [C Language].

POSIX.1q: 2000

IEEE Std 1003.1q-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 7: Tracing [C Language].

POSIX.2b

P1003.2b, Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment

POSIX.2d: 1994

IEEE Std 1003.2d: 1994, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment 1: Batch Environment.

SVID, Issue 1

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 1; Morristown, NJ, UNIX Press, 1985.

SVID, Issue 2

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 2; Morristown, NJ, UNIX Press, 1986.

SVID, Issue 3

American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 3; Morristown, NJ, UNIX Press, 1989.

XNS, Issue 5.2

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

X/Open Curses, Issue 4, Version 2

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

X/Open Curses, Issue 7

Technical Standard, November 2009, X/Open Curses, Issue 7 (ISBN: 1-931624-83-6) C094), published by The Open Group.

The Single UNIX Environment

The Single UNIX Specification programming environment provides a broad-based functional set of interfaces to support the porting of existing UNIX applications and the development of new applications. The environment also supports a rich set of tools for application development.

This chapter describes the overall environment, programming, and utility considerations, and some of the global issues such as internationalization support, environment variables, and file access. It describes how the environment is organized and where to look for detailed information in the Single UNIX Specification. References give the section of the Single UNIX Specification in which more detailed explanations and definitions can be found.

1.1 Introduction

Many names have been applied to the work that has culminated in the Single UNIX Specification and the accompanying UNIX certification program. It began as the Common API Specification, became Spec 1170, and is now in its latest iteration the Single UNIX Specification, Version 4, published in a number of The Open Group Technical Standards, the core of which are also POSIX.1-2008.

The Single UNIX Specification uses The Open Group Base Specifications, Issue 7 documentation as its core. The documentation is structured as follows:

- The Base Specifications, Issue 7, composed of:
 - Base Definitions, Issue 7 (XBD7)
 - System Interfaces, Issue 7 (XSH7)
 - Shell and Utilities, Issue 7 (XCU7)
 - Rationale, Issue 7, (XRAT7) (Informative)
- X/Open Curses, Issue 7

One thing that becomes apparent working with the Single UNIX Specification is its focus on application development. The Single UNIX Specification is similar to the User's and Programmer's Reference Manuals on Berkeley or System V systems. Matters of system management are not part of this specification. Directory organization is not discussed beyond the simple few directories and devices that applications generally use. User management discussions do not appear. There is no discussion of such files as **/etc/passwd** or **/etc/groups**, since an application's access to the information traditionally kept in these files is through programmatic interfaces such as *getpwnam()* and *getgrnam()*. Processes have *appropriate privileges*, and there is no concept of the "superuser" or "root".

1.2 Interface Counts

There are now 1833 interfaces defined in the Single UNIX Specification, Version 4. The table below gives the breakdown for each volume of the Single UNIX Specification. Also see the interface tables in [Chapter 7](#) through [Chapter 10](#) for detailed lists of the interfaces.

XSH	XCU	XBD	XCURSES	Total
1191	174	82	386	1833

Table 1-1 Interface Count

1.3 Formal Standards Alignment

The Single UNIX Specification supports formal standards developed for applications portability. The following source code portability standards lie at the core of the Single UNIX Specification:

- POSIX.1-2008

(This is technically identical to the Base Specifications, Issue 7; they are one and the same document.)

- The ISO/IEC 9899: 1999 standard

The Single UNIX Specification fully aligns with these standards. The Single UNIX Specification functional extensions beyond the required POSIX base functionality are identified by the X/Open System Interfaces option, denoted XSI, and by mandating certain other POSIX options; for example, File Synchronization. Where there are multiple interfaces to accomplish some task, the standards-based interfaces are clearly identified as the preferred way of doing things to support future portability.

1.4 Portability Codes

While great care has been taken to align the Single UNIX Specification with formal standards, it is still a superset specification that extends functionality or perhaps presents a more exact (and restrictive) definition. When this occurs, the text in the Single UNIX Specification is clearly marked by shading and a portability code appears in the margin (sometimes referred to as a margin code or option code). In this version, the number of portability codes has reduced, as the functionality associated with various old options has been mandated.

Programmers need to take care when using functionality that appears in a shaded area if they are developing applications that need to be maximally portable or portable beyond UNIX certified systems. For example, if functionality is marked with XSI in the margin, it will be available on all UNIX certified systems, but may not be available on systems only supporting the base POSIX.1 requirements. Alternatively, an application may depend on the exact format of output from a particular utility whose output format is incompletely specified, as indicated by shading and OF marked in the margin. It is likely that an application is developed with a particular platform, or at least a well-defined set of platforms in mind. These codes are exceptionally useful to warn a developer of areas of potential problems.

There are 45 margin codes defined in total in XBD, Section 2.1.6, Options. 40 of these reflect optional features defined within the POSIX base standard, six of which are mandatory in the Single UNIX Specification (FSC, TSA, TSH, TSS, UP, and XSI). One of the codes—MC1—is a

shorthand notation for a permutation of certain options. Three of the codes are special codes to denote other portability warnings, these being OB (obsolescent), OH (optional header), and OF (output incompletely specified).

All the codes including those of the POSIX options are listed below together with an indication of their status within the Single UNIX Specification:

ADV	Advisory Information
	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Advanced Realtime Option Group.
BE	Batch Environment Services and Utilities
	This identifies utilities and services that are optional in the Single UNIX Specification. They are defined in XCU.
CD	C-Language Development Utilities
	This identifies a set of utilities that are optional in the Single UNIX Specification. They are defined in XCU.
CPT	Process CPU-Time Clocks
	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Advanced Realtime Option Group.
CX	Extension to the ISO C Standard
	Extensions beyond the ISO C standard are so marked. These extensions are mandatory on all systems supporting POSIX.1 and the Single UNIX Specification. This is included since the Single UNIX Specification defers to the ISO C standard, thereby allowing the extensions to be identified.
FD	FORTRAN Development Utilities
	This identifies a set of utilities that are optional in the Single UNIX Specification. They are defined in XCU.
FR	FORTRAN Runtime Utilities
	This identifies a set of utilities that are optional in the Single UNIX Specification. They are defined in XCU.
FSC	File Synchronization
	This identifies interfaces and additional semantics that are optional within the POSIX base standard but mandatory in the Single UNIX Specification.
IP6	IPV6
	This identifies additional semantics for networking interfaces relating to support of IP Version 6 that are optional in the Single UNIX Specification.
MC1	Non-Robust Mutex Priority Protection or Non-Robust Mutex Priority Inheritance or Robust Mutex Priority Protection or Robust Mutex Priority Inheritance
	This is a special margin code used as a shorthand notation.
ML	Process Memory Locking
	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Realtime Option Group.

MLR	Range Memory Locking	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Realtime Option Group.
MON	Monotonic Clock	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Advanced Realtime Option Group.
MSG	Message Passing	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Realtime Option Group.
MX	IEC 60559 Floating-Point Option	This identifies additional semantics within Math interfaces that are optional in the Single UNIX Specification. Indicates support of the IEC 60559: 1989 standard floating-point.
OB	Obsolescent	Features marked as obsolescent are portable to all Single UNIX Specification platforms, but may be withdrawn in a future issue. This functionality should be avoided.
OF	Output Format Incompletely Specified	The format of a utility's output is incompletely specified, and therefore cannot be processed consistently by another application across multiple platforms.
OH	Optional Header	Certain headers are not required to be included in a source module on XSI-conforming systems, although the POSIX base standard may require it. These headers are marked in the SYNOPSIS sections.
PIO	Prioritized Input and Output	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification.
PS	Process Scheduling	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Realtime Option Group.
RPI	Robust Mutex Priority Inheritance	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. This is new functionality added in this version of the specification.
RPP	Robust Mutex Priority Protection	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. This is new functionality added in this version of the specification.
RS	Raw Sockets	This identifies additional semantics for sockets that are optional in the Single UNIX Specification.
SD	Software Development Utilities	This identifies a set of utilities that are optional in the Single UNIX Specification. They are defined in XCU.

SHM	Shared Memory Objects	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Realtime Option Group.
SIO	Synchronized Input and Output	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Realtime Option Group.
SPN	Spawn	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Advanced Realtime Option Group.
SS	Process Sporadic Server	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Advanced Realtime Option Group.
TCT	Thread CPU-Time Clocks	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Advanced Realtime Threads Option Group.
TEF	Trace Event Filter	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Tracing Option Group.
TPI	Non-Robust Mutex Priority Inheritance	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Realtime Threads Option Group.
TPP	Non-Robust Mutex Priority Protection	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Realtime Threads Option Group.
TPS	Thread Execution Scheduling	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Realtime Threads Option Group.
TRC	Trace	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Tracing Option Group.
TRI	Trace Inherit	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Tracing Option Group.
TRL	Trace Log	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Tracing Option Group.
TSA	Thread Stack Address Attribute	This identifies semantics that are optional within the POSIX base standard but mandatory in the Single UNIX Specification.

TSH	Thread Process-Shared Synchronization
	This identifies interfaces and additional semantics that are optional within the POSIX base standard but mandatory in the Single UNIX Specification.
TSP	Thread Sporadic Server
	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Advanced Realtime Threads Option Group.
TSS	Thread Stack Size Attribute
	This identifies semantics that are optional within the POSIX base standard but mandatory in the Single UNIX Specification.
TYM	Typed Memory Objects
	This identifies interfaces and additional semantics that are optional in the Single UNIX Specification. They are part of the Advanced Realtime Option Group.
UP	User Portability Utilities
	This identifies interfaces and additional semantics in XCU that are optional within the POSIX base standard but mandatory in the Single UNIX Specification. These represent interactive utilities or semantics within utilities.
UU	UUCP Utilities
	This identifies a set of utilities that are optional in the Single UNIX Specification. They are defined in XCU.
XSI	X/Open System Interfaces
	This identifies interfaces and additional semantics that are optional within the POSIX base standard but mandatory in the Single UNIX Specification. This functionality is required to be present on all UNIX certified systems.
XSR	XSI STREAMS
	This identifies interfaces and additional semantics related to the support of STREAMS. This functionality is optional within this version of the Single UNIX Specification.

Where a portability code applies to an entire function or utility, the SYNOPSIS section of the corresponding reference page is shaded and marked with the margin code. Refer to XBD, Section 1.7, Portability for more information.

1.5 Option Groups

The Single UNIX Specification includes a set of profiling options, allowing larger profiles of the options of the Base standard. The Option Groups within the Single UNIX Specification are defined within XBD, Section 2.1.5.2, XSI Option Groups.

The Single UNIX Specification contains the following Option Groups:

- **Encryption**, covering the functions *crypt()*, *encrypt()*, and *setkey()*
- **Realtime**, covering the functions from the IEEE Std 1003.1b-1993 Realtime extension.
- **Realtime Threads**, covering threads-related functions that are also related to realtime functionality.

- **Advanced Realtime**, covering some of the non-threads-related functions originally from IEEE Std 1003.1d-1999 and IEEE Std 1003.1j-2000.
- **Advanced Realtime Threads**, covering some of the threads-related functions originally from IEEE Std 1003.1d-1999 and IEEE Std 1003.1j-2000.
- **Tracing**, covering the functionality originally from IEEE Std 1003.1q-2000. This functionality has been marked obsolescent in this version of the Single UNIX Specification.
- **XSI STREAMS**, covering the functionality and interfaces related to STREAMS, a uniform mechanism for implementing networking services and other character-based I/O as described in XSH, Section 2.6, STREAMS. This functionality has been marked obsolescent in this version of the Single UNIX Specification.

1.6 Common Directories and Devices

The Single UNIX Specification describes an applications portability environment, and as such defines a certain minimal set of directories and devices that applications regularly use. The following directories are defined:

- /** The root directory of the file system.
- /dev** Contains the devices **/dev/console**, **/dev/null**, and **/dev/tty**.
- /tmp** A directory where applications can create temporary files.

The directory structure does not cross into such system management issues as where user accounts are organized or software packages are installed. Refer to XBD, Section 10.1, Directory Structure and Files for more information.

XBD, Chapter 10, Directory Structure and Devices also defines the mapping of <control>-*char* sequences to control character values, and associated requirements on system documentation.

1.7 Environment Variables

When a program begins, an environment is made available to it. The environment consists of strings of the form *name=value*, where *name* is the name associated with the environment variable, and its value is represented by the characters in *value*. UNIX systems traditionally pass information to programs through the environment variable mechanism. The Single UNIX Specification uses only uppercase characters, digits, and underscores to name environment variables, reserving for applications the name space of names containing lowercase characters.

A number of utilities and functions defined in the Single UNIX Specification use environment variables to modify their behavior. The ENVIRONMENT VARIABLES section of a utility's reference page describes any appropriate environment variables. Quite a number of environment variables (listed below) modify the behavior of more than a single utility.

ARFLAGS	IFS	MAILPATH	PS1
CC	LANG	MAILRC	PS2
CDPATH	LC_ALL	MAKEFLAGS	PS3
CFLAGS	LC_COLLATE	MAKESHELL	PS4
CHARSET	LC_CTYPE	MANPATH	PWD
COLUMNS	LC_MESSAGES	MBOX	RANDOM
DATMSK	LC_MONETARY	MORE	SECONDS
DEAD	LC_NUMERIC	MSGVERB	SHELL
EDITOR	LC_TIME	NLSPATH	TERM
ENV	LDFLAGS	NPROC	TERMCAP
EXINIT	LEX	OLDPWD	TERMINFO
FC	LFLAGS	OPTARG	TMPDIR
FCEDIT	LINENO	OPTERR	TZ
FFLAGS	LINES	OPTIND	USER
GET	LISTER	PAGER	VISUAL
GFLAGS	LOGNAME	PATH	YACC
HISTFILE	LPDEST	PPID	YFLAGS
HISTORY	MAIL	PRINTER	
HISTSIZE	MAILCHECK	PROCLANG	
HOME	MAILER	PROJECTDIR	

1.8 YACC Grammars as Specifications

The Single UNIX Specification describes certain functionality and “little” languages with *yacc* grammars. The *yacc* utility (Yet-Another-Compiler-Compiler) is used to build language parsers, and the syntax of the *yacc* input language is designed for describing languages. Because of this, and its familiarity in the community, it has been used in the Single UNIX Specification to describe the shell (XCU, Chapter 2, Shell Command Language), the *awk* language (XCU, Chapter 4, Utilities, *awk*), the *bc* language (XCU, Chapter 4, Utilities, *bc*), the locale definition language (XBD, Chapter 7, Locale), basic and extended regular expressions (XBD, Chapter 9, Regular Expressions), and *terminfo* source format (X/Open Curses, Chapter 7, Terminfo Source Format).

These grammars are representations of the languages they describe, and a number of caveats apply:

- They are not complete input grammars to *yacc* itself.
- They are not based on any existing implementations, nor have they been tested as such.
- Were these partial grammars to be completed, there are no guarantees that they are the most efficient method of defining the respective languages for *yacc* to process.

Refer to XCU, Chapter 1 for more information.

1.9 Regular Expressions

Both Basic Regular Expressions (BREs) and Extended Regular Expressions (EREs) are described in XBD, Chapter 9, Regular Expressions and all of the utilities and interfaces that use regular expressions refer back to this definition.

Basic regular expressions:

csplit, ed, ex, expr, grep, more, nl, pax, sed, vi

Extended regular expressions:

awk, grep -E, lex

The functions *regcomp()* and *regexexec()* in XSH, Chapter 3, System Interfaces implement regular expressions as defined in the Single UNIX Specification.

1.10 File Access

The Single UNIX Specification describes the interaction of symbolic links that may effect the file access in the file system. The behavior of symbolic links is fully specified with respect to their creation and use through the relevant XSH interfaces, such as *lchown()*, *lstat()*, *readlink()*, *realpath()*, and *symlink()*, and pathname resolution (XBD, Chapter 4, General Concepts). The behavior is also specified with respect to the utilities described in XCU.

1.11 Programming Environment

The Single UNIX Specification and UNIX certified systems are tools for developing portable applications and for porting existing applications that were originally developed to run on UNIX systems. XSH defines the C-language programming environment, and the syntax and semantics of the interfaces. Feature test macros, name space issues, and the program interaction with the operating system are all described in the opening chapters of XSH. The following introductions to these topics include references and additional explanations to orient an application developer with the information presented in XSH.

1.11.1 C-Language Support

The programming interfaces in XSH are described in C-language syntax, as defined in the ISO C standard, and presume a C-language compilation environment. (Implementations may make the functionality available through other programming languages, but this is not covered by the Single UNIX Specification.)

For an implementation to be a conforming UNIX certified system, it must support the ISO C standard. Implementations may additionally support the X/Open Common Usage C dialect as a migration strategy from previous XPG3 environments. X/Open Common Usage C is defined in Programming Languages, Issue 3, Chapters 1-4, and essentially refers to the C Language before the 1989 ANSI C standard.

XCU defines *c99* as the interface to the C compilation environment. The *c99* interface is an interface to the standard C99 compiler. The *c89* and *cc* utilities are not defined in this version of the Single UNIX Specification, although implementations may additionally support them for

backwards-compatibility.

1.11.2 Feature Test Macros and Name Space Issues

There are a number of tasks that must be done to effectively make the interface environment available to a program. One or more C-language macros, referred to as *feature test macros*, must be defined before any headers are included. These macros might more accurately be referred to as header configuration macros, as they control what symbols will be exposed by the headers. The macro `_XOPEN_SOURCE` must be defined to a value of 700 to make available the functionality of the Single UNIX Specification, Version 4. With respect to POSIX base functionality covered by the Single UNIX Specification, this is equivalent to defining the macro `_POSIX_C_SOURCE` to be 200809L.

Use of the `_XOPEN_SOURCE` macro should not be confused with indicator macros associated with options and Option Groups, such as `_XOPEN_UNIX`, which are defined by the implementation in `<unistd.h>`.

In the first case (feature test macro `_XOPEN_SOURCE` defined by the application), the application is announcing to the implementation its desire to have certain symbols exposed in standard headers. The implementation tests the value of the macro to determine what features have been requested. In the second case (indicator macro `_XOPEN_UNIX` defined by the implementation in `<unistd.h>`), the implementation is announcing to the application what functionality it supports. The application tests the macro value to determine whether the implementation supports the functionality it wants to use (the XSI option in this example).

The UNIX system name space is well defined. All identifiers defined in the Single UNIX Specification appear in the headers described in XBD, Chapter 13, Headers (with the exception of *environ*). Tables in XSH, Section 2.2, The Compilation Environment clearly describe which identifier prefixes and suffixes are reserved for the implementation, which identifier macro prefixes may be used by the application programmer providing appropriate `#undef` statements are used to prevent conflicts, and which identifiers are reserved for external linkage.

1.11.3 Error Numbers

Each interface reference page lists in the ERRORS section possible error returns that may be tested either in *errno* or in the function return value upon the unsuccessful completion of a function call. Each error return has a symbolic name (defined as a manifest constant in `<errno.h>`) which should always be used by the portable application, as the actual error values are unspecified. All of the error names are listed in XSH, Section 2.3, Error Numbers along with additional relevant information.

1.11.4 Signal Concepts

The *signal()* function defined by the ISO C standard has shortcomings that make it unreliable for many application uses on some implementations. The Single UNIX Specification defines a reliable signal mechanism which applications should use instead. XSH, Section 2.4, Signal Concepts discusses signal generation and delivery, signal actions, async-signal-safe functions, and interruption of functions by signals.

1.11.5 Standard I/O Streams

When a program starts, it has three I/O streams associated with it, namely *standard input* (for reading conventional input), *standard output* (for writing conventional output), and *standard error* (for writing diagnostic output). These streams are already open for the process, and ready for I/O.

The mechanics of stream I/O, buffering, relationships to file descriptors, and inheritance across process creation are all discussed in XSH, Section 2.5, Standard I/O Streams. New in this version of the Single UNIX Specification are facilities for associating a standard I/O stream with a memory buffer instead of a file.

1.11.6 STREAMS

STREAMS is a method of implementing network services and other character-based input/output mechanisms, with the STREAM being a full-duplex connection between a process and a device. STREAMS provides direct access to protocol modules, and optional protocol modules can be interposed between the process-end of the STREAM and the device-driver at the device-end of the STREAM. Pipes can be implemented using the STREAMS mechanism, so they can provide process-to-process as well as process-to-device communications.

XSH, Section 2.6, STREAMS introduces STREAMS I/O, the message types used to control them, an overview of the priority mechanism, and the interfaces used to access them. In this version of the specification, STREAMS functionality has been marked obsolescent.

1.11.7 XSI Interprocess Communication

The Single UNIX Specification describes a set of interprocess communications (IPC) primitives, namely message queues, semaphores, and shared memory. These functions are all derived from the SVID.

General information that is shared by all three mechanisms is described in XSH, Section 2.7, XSI Interprocess Communication. There, the common permissions mechanism is briefly introduced, describing the mode bits and how they are used to determine whether or not a process has access to read or write/alter the appropriate instance of one of the IPC mechanisms. All other relevant information is contained in the reference pages themselves.

1.11.8 Realtime

The Single UNIX Specification includes realtime functionality to support the source portability of applications with realtime requirements. Realtime is discussed in XSH, Section 2.8, Realtime. This section includes an overview of the functional areas: semaphores, process memory locking, memory mapped files and shared memory objects, priority scheduling, realtime signals, timers, interprocess communication, synchronous input/output, and asynchronous input/output.

1.11.9 Threads

XSH describes functionality to support multiple flows of control, called *threads*, within a process. Threads are discussed in XSH, Section 2.9, Threads, which includes an overview of the supported interfaces, threads implementation models, thread mutexes, thread attributes, thread scheduling, thread cancelation, thread read-write locks, and application-managed thread stacks.

1.11.10 Sockets

UNIX certified systems support UNIX domain sockets for process-to-process communication in a single system and network sockets using Internet protocols based on IPv4, and may also support raw sockets and network sockets using Internet protocols based on IPv6.

XSH Section 2.10, Sockets, discusses all aspects of UNIX domain sockets and network sockets, including socket types, addressing, protocols, and socket options.

1.11.11 General Terminal Interface

The general terminal interface is described in the Single UNIX Specification in XBD, Chapter 11, General Terminal Interface, providing a mechanism to control asynchronous communications ports. It is left to implementations as to whether or not they support network connections and synchronous communications ports.

While all of the interface details are contained in XSH, the mechanics of the terminal interface with respect to process groups, controlling terminals, input and output processing, input, output, and control modes, and special characters are described in XBD, Chapter 11, General Terminal Interface.

This interface should not be confused with the X/Open Curses interface that provides a terminal-independent way to update character screens.

1.11.12 How to Read an XSH Reference Page

Each reference page in XSH has a common layout of sections describing the interface. (Function interface descriptions in X/Open Curses follow the same layout.) This layout is similar to the manual page or “man” page format shipped with most UNIX systems, and each interface has SYNOPSIS, DESCRIPTION, RETURN VALUE, and ERRORS sections. These are the four sections that relate to conformance.

Additional sections contain considerable extra information for the application developer. The EXAMPLES sections provide source code examples of how to use certain interfaces. The APPLICATION USAGE sections provide additional caveats, issues, and recommendations to the developer. The SEE ALSO sections contain useful pointers to related interfaces and headers that a developer may wish to also read.

The FUTURE DIRECTIONS sections act as pointers to related work that may impact on the interface in the future, and often cautions the developer to architect the code to account for a change in this area. (A FUTURE DIRECTIONS section expresses current thinking and should not be considered a commitment to adopt the feature or interface in the future.)

The RATIONALE sections include historical information about an interface and why features were included or discarded in the definition.

The CHANGE HISTORY sections describe when the interface was introduced, and how it has

changed. This information can be useful when porting existing applications that may reflect earlier implementations of the interface.

Option Group labels in the reference page headers, and portability shading and margin marks are features already described in this document; they appear on the reference pages to guide an application developer when deciding how best an interface should be used. Refer to XSH, Section 1.2, Format of Entries for information on the exact layout.

1.12 Commands and Utilities Environment

The Single UNIX Specification describes 174 utilities supported on UNIX certified systems. These utilities provide a rich environment for building shell script applications, supporting program development (the C Language in particular), and providing a user portability environment.

The shell command language, symbolic links, file format notation, utility reference page layouts, and guidelines, are all introduced in this section. The following introductions to these topics include references and additional explanations to orient an application developer with the information presented in XCU.

1.12.1 Shell Command Language

The shell is a powerful and flexible programming language. A considerable number of utilities on early UNIX systems were actually shell programs.

The Single UNIX Specification shell is the standard POSIX shell. This shell is for the most part based on the Bourne shell with features from the KornShell, *ksh*.

The shell command language is defined in its entirety in XCU, Chapter 2, Shell Command Language. This is a very strict definition of the shell. Token recognition, word expansions, simple and compound commands, the shell grammar, the execution environment, and special built-ins are a few of the topics covered. It is not a guide to writing or porting shell scripts.

1.12.2 Symbolic Links

The Single UNIX Specification includes support for symbolic links in the file system. The programmatic interfaces that manipulate symbolic links (or *symlinks*) are all well defined in XSH and XCU, and *symlink* concepts with respect to issues like pathname resolution are discussed in XBD, Chapter 4, General Concepts. A UNIX certified system supports *symlinks*, and application programs may make use of them.

1.12.3 File Format Notation

Sections in the utility reference pages often require the expected input used by the utility or output it generates to be described. Additionally, information files used or created sometimes require description. The method used throughout XCU is a format description plus its arguments, similar to that used by the *printf()* function. These file format specifications are presented as:

```
"<format>" [ , <arg1> , <arg2> , ... , <argn> ]
```

The *format* specifier contains the format string a programmer might use to write the data. The

specifiers should not be considered format strings that could be directly used in a call to `scanf()`. The conversion specifications in the format string are what would be expected by a developer familiar with the `printf()` interface. Refer to XBD, Chapter 5, File Format Notation for more information.

1.12.4 How to Read an XCU Reference Page

Each reference page in XCU has a common layout of sections describing the interface. This layout, while similar to the manual page or “man” page format shipped with most UNIX systems, offers a more detailed view of the utility’s description.

As well as the SYNOPSIS and DESCRIPTION sections, each interface has OPTIONS, OPERANDS, STDIN (standard input format), INPUT FILES, ENVIRONMENT VARIABLES, ASYNCHRONOUS EVENTS (what signals are caught and the consequence of receiving signals), STDOUT (standard output format), STDERR, and OUTPUT FILES sections.

An EXTENDED DESCRIPTION will be used if the utility has a particularly long description; for example, if it supports its own language (`awk`), or provides a considerable number of subcommands (`mailx`).

Utilities generally return 0 upon successful completion, and a failed status as greater than 0. The EXIT STATUS sections specify this, but will also describe if particular values are returned in certain circumstances. In general, an application should be written to test for successful completion, rather than specific error returns.

The CONSEQUENCE OF ERRORS section describes what happens to such items as open files, process state, and the environment, should errors occur.

As with the XSH reference pages, additional sections contain considerable extra information for the application developer, and include EXAMPLES, APPLICATION USAGE, FUTURE DIRECTIONS, RATIONALE, SEE ALSO, and CHANGE HISTORY sections. The defaults for these sections, and additional detail about what each section specifies, are covered in XCU, Section 1.4, Utility Description Defaults.

1.13 Terminal Interfaces Environment

The Single UNIX Specification includes X/Open Curses. These interfaces provide a terminal-independent character screen update method. The functionality includes:

- Multibyte and wide-character support
- Color terminal support
- Wide and non-spacing character generalizations such that multi-column display characters and non-spacing characters are supported

New interfaces over Issue 3 are marked with ENHANCED CURSES in the reference page header, and shading and a portability code is used (EC) to indicate in the introductory chapters and reference pages the extended features.

Applications using the interfaces from X/Open Curses need to define the `_XOPEN_SOURCE` macro to be 700 prior to including any headers. This feature test macro ensures the appropriate name space is exposed in the headers.

On UNIX certified systems, the `c99` compiler recognizes the additional `curses` library option-argument for the `-l` option.

There are four X/Open Curses utilities—*infocmp*, *tic*, *tput*, and *untic*—which are new in this version of the Single UNIX Specification.

1.14 Internationalization

There is a rich set of interfaces in the Single UNIX Specification to support internationalized applications development. Internationalization refers to developing an application without prior knowledge of the language, cultural information, or character set encoding scheme that will be used in the run-time environment. The application responds accordingly at run time for cultural or locale-specific conditions. The term “internationalization” is often shortened to simply “I18N”, for “I”—18 letters (internationalization)—“N”.

Localization is the process of establishing a base of cultural and codeset data on a system such that it can be accessed. It is the method by which *locales* are created in such a way that internationalized programs can access the relevant information.

There are a number of factors that need to be considered when structuring an application to support multiple cultures:

- Cultural-specific data (for example, date, number, and monetary formats) needs to be defined in a locale, with the appropriate tools and interfaces available for creating and accessing the information.
- Provision for handling text strings in programs, and managing catalogs of these strings for multiple natural languages, needs to be made.
- Multibyte and wide-character handling interfaces are required to manage textual data that is not represented in single byte per character formats.
- Codeset conversion tools need to be supported.

All of these requirements are met in the Single UNIX Specification. Utilities exist to support the definition and display of locales (*localedef* and *locale*). Programming interfaces exist to access the locale-specific data, and others have been extended such that all the checking and manipulations done with characters and strings can be done with wide characters and wide-character strings. Interfaces to access message string catalogs (*catopen()*, *catgets()*, and *catclose()*) and a tool to define them (*genccat*) exist. Codeset conversions can be done at the utility (*iconv*) and program level (*iconv_open()*, *iconv()*, and *iconv_close()*). New in this version of the Single UNIX Specification are facilities for handling multiple concurrent locales (*newlocale()*, *duplocale()*, *uselocale()*, and variants of existing functions with names ending *_l*; for example, *isalnum_l()*).

Much of the I18N information is spread throughout the Single UNIX Specification, with specific interface and utility descriptions appearing in XSH and XCU, respectively. Introductions to character and codeset issues and the locale definition language appear in XBD, Chapter 6, Character Set, and Chapter 7, Locale, respectively.

The Single UNIX Specification, Version 4

This chapter gives an overview of the different parts that comprise the Single UNIX Specification, Version 4 and how they are organized. The Single UNIX Specification, Version 4 is made up of two documents: the Base Specifications, Issue 7 and X/Open Curses, Issue 7. The Base Specifications, Issue 7 comprise four volumes: Base Definitions, System Interfaces, Shell and Utilities, and Rationale.

2.1 Base Definitions (XBD)

XBD is part of the Base Specifications, Issue 7. XBD provides common definitions for the Base Specifications of the Single UNIX Specification; therefore, readers should be familiar with it before using the other parts of the Single UNIX Specification. The presence of this volume reduces duplication in the other related parts of the Single UNIX Specification and ensures consistent use of terminology.

This volume is structured as follows:

- Chapter 1 is an introduction which includes the scope of the Base Specifications, and the scope of the changes made in this version. Normative references, terminology, and portability codes used throughout the Base Specifications are included in this chapter.
- Chapter 2 defines the conformance requirements, both for implementation and application conformance. For implementation conformance, this includes documentation requirements, conformance definitions for the core POSIX subset, conformance definitions for systems conforming to the Single UNIX Specification (denoted as XSI conformance), and option groups.
- Chapter 3 contains the general terms and definitions that apply throughout the Base Specifications.
- Chapter 4 describes general concepts that apply throughout the Base Specifications.
- Chapter 5 describes the notation used to specify file input and output formats in XBD and XCU.
- Chapter 6 describes the portable character set and the process of character set definition.
- Chapter 7 describes the syntax for defining internationalization locales as well as the POSIX locale provided on all systems.
- Chapter 8 describes the use of environment variables for internationalization and other purposes.
- Chapter 9 describes the syntax of pattern matching using regular expressions employed by many utilities and matched by the *regcomp()* and *regexexec()* functions. Both Basic Regular Expressions (BREs) and Extended Regular Expressions (EREs) are described in this chapter.

- Chapter 10 describes files and devices found on all systems and their semantics. For example, the device **/dev/null** is an infinite data source and data sink.
- Chapter 11 describes the asynchronous terminal interface for many of the functions in XSH and the *stty* utility in XCU.
- Chapter 12 describes the policies for command line argument construction and parsing. It contains the utility argument syntax used throughout XCU, and also utility syntax guidelines for naming of utilities and the specification of their options, option-arguments, and operands.
- Chapter 13 defines the contents of headers which declare the functions and global variables, and define types, constants, macros, and data structures that are needed by programs using the services provided by the system interfaces defined in XSH. These are in the form of reference pages and are organized alphabetically.

2.2 System Interfaces (XSH)

XSH is part of the Base Specifications, Issue 7. XSH describes a set of system interfaces offered to application programs by systems conformant to this volume of the Single UNIX Specification. Readers are expected to be experienced C language programmers, and to be familiar with the XBD volume.

This volume is structured as follows:

- Chapter 1 explains the relationship of XSH to other formal standards and describes the format of the reference pages that comprise most of this volume.
- Chapter 2 contains important concepts, terms, and caveats relating to the rest of XSH. This includes information on the compilation environment, the name space, definitions of error numbers, signal concepts, standard I/O streams, STREAMS, XSI IPC, realtime, threads, sockets, tracing, and data types.
- Chapter 3 defines the functional interfaces to systems conformant to this volume of the Single UNIX Specification. These are in the form of reference pages and are organized alphabetically.

2.3 Shell and Utilities (XCU)

XCU¹ is part of the Base Specifications, Issue 7. XCU describes the shell and utilities that are available to application programs on systems conformant to this volume of the Single UNIX Specification. Readers are expected to be familiar with the XBD volume.

This volume is structured as follows:

- Chapter 1 explains the relationship of XCU to other formal standards, including the ISO C standard and also XSH. It also describes the utility limits, grammar conventions, defaults used by the utility descriptions, considerations for utilities in support of large files, and requirements for built-in utilities.

1. The acronym "XCU" derives from an earlier version of this volume of the specification which was called "Commands and Utilities".

- Chapter 2 describes the command language—that is, the shell command language interpreter—used in systems conformant to the Single UNIX Specification. It also includes reference pages for the special built-in utilities, organized alphabetically.
- Chapter 3 describes a set of services and utilities that are implemented on systems supporting the Batch Environment option.
- Chapter 4 consists of reference pages for all utilities—other than the special built-in utilities described in Chapter 2—available on systems conforming to the Single UNIX Specification. These are in the form of reference pages and are organized alphabetically.

2.4 Rationale (XRAT)

XRAT is part of the Base Specifications, Issue 7. XRAT has been published to assist in the process of review and understanding of the main text. It contains historical information concerning the contents of the Base Specifications, Issue 7 and why features were included or discarded by the standard developers. It also contains notes of interest to application programmers on recommended programming practices, emphasizing the consequences of some aspects that may not be immediately apparent.

XRAT is organized in parallel to the normative volumes of the Base Specification, with a separate appendix (A, B, and C) for each of the three normative volumes of the document: XBD, XSH, and XCU, respectively. In addition, two additional appendices are included: Appendix D, Portability Considerations and Appendix E Subprofiling Considerations. The Portability Considerations appendix includes a report on the perceived user requirements for the Base Specifications and how the facilities provided satisfy those requirements, together with guidance to writers of profiles on how to use the configurable options, limits, and optional behavior. The Subprofiling Considerations appendix satisfies the requirement that the document address subprofiling. This contains an example set of subprofiling options.

2.5 X/Open Curses (XCURSES)

XCURSES is not part of the Base Specifications, Issue 7, but is a separate Open Group Technical Standard. XCURSES describes a set of interfaces providing a terminal-independent method of updating character screens that are available to application programs on systems conformant to this part of the Single UNIX Specification.

This document is structured as follows:

- Chapter 1 introduces Curses and gives an overview of the changes made in this version. This chapter also defines the requirements for conformance to this document and shows the generic format followed by interface definitions in Chapter 4.
- Chapter 2 describes the relationship between Curses and the C language, the compilation environment, and the X/Open System Interface (XSI) operating system requirements. It also defines the effect of the interface on the name space for identifiers and introduces the major data types that the interfaces use.
- Chapter 3 gives an overview of Curses. It discusses the use of some of the key data types and gives general rules for important common concepts such as characters, renditions, and window properties. It contains general rules for the common Curses operations and operating modes. This information is implicitly referenced by the interface definitions in Chapter 4. The chapter explains the system of naming the Curses functions and presents

a table of function families. Finally, the chapter contains notes regarding use of macros and restrictions on block-mode terminals.

- Chapter 4 defines the Curses functional interfaces.
- Chapter 5 defines the contents of headers which declare the functions and global variables, and define types, constants, macros, and data structures that are needed by programs using the services provided by the interfaces defined in Chapter 4.
- Chapter 6 defines the Curses utilities.
- Chapter 7 discusses the *terminfo* database which Curses uses to describe terminals. The chapter specifies the source format of a *terminfo* entry using a formal grammar, an informal discussion, and an example. Boolean, numeric, and string capabilities are presented in tabular form.
- Appendix A discusses the use of these capabilities by the writer of a *terminfo* entry to describe the characteristics of the terminal in use.
- The Glossary contains normative definitions of terms used in the document.

System Interfaces

The Single UNIX Specification covers a broad spectrum of functionality, providing a rich environment for applications portability and portable applications development. This chapter covers the C language programming interfaces that are the system interfaces, describing the scope of the interfaces, an overview of the changes in this version, and lists of the new functions. It also groups the interfaces into categories to give the reader an understanding of what facilities are available.

3.1 Base Documents

The system interfaces provided in XSH, Issue 7 are drawn from the following base documents:

- IEEE Std 1003.1, 2004 Edition (POSIX-1) (incorporating IEEE Std 1003.1-2001, IEEE Std 1003.1-2001/Cor 1-2002, and IEEE Std 1003.1-2001/Cor 2-2004)
- The Open Group Technical Standard, 2006, Extended API Set Part 1
- The Open Group Technical Standard, 2006, Extended API Set Part 2
- The Open Group Technical Standard, 2006, Extended API Set Part 3
- The Open Group Technical Standard, 2006, Extended API Set Part 4
- ISO/IEC 9899:1999, Programming Languages — C (including ISO/IEC 9899:1999/Cor.1:2001(E), ISO/IEC 9899:1999/Cor.2:2004(E), and ISO/IEC 9899:1999/Cor.3(E))

3.2 Overview of Changes

The following sections describe changes made to XSH since Issue 6. [Chapter 11](#) (on page 101) details the technical changes that have been made to the system interfaces from Issue 6.

[Chapter 11](#) (on page 101) also details the changes to the POSIX base requirements.

Changes from Issue 6 to Issue 7

The following list summarizes the major changes that were made in XSH from Issue 6 to Issue 7:

- The Open Group Technical Standard, 2006, Extended API Set Part 1 is incorporated.
- The Open Group Technical Standard, 2006, Extended API Set Part 2 is incorporated.
- The Open Group Technical Standard, 2006, Extended API Set Part 3 is incorporated.

- The Open Group Technical Standard, 2006, Extended API Set Part 4 is incorporated.
- Existing functionality is aligned with ISO/IEC 9899: 1999, Programming Languages — C, ISO/IEC 9899: 1999/Cor.2: 2004(E).
- Austin Group defect reports, IEEE Interpretations against IEEE Std 1003.1, and responses to ISO/IEC defect reports against ISO/IEC 9945 are applied.
- The Open Group corrigenda and resolutions are applied.
- Issues arising from ISO TR 24715: 2006, Conflicts between POSIX and the LSB have been addressed.
- Features, marked legacy or obsolescent in the base document, have been considered for removal in this version. The following table lists the removed functions and symbols:

Removed Functions and Symbols in Issue 7		
<i>bcmp()</i>	<i>gethostbyaddr()</i>	<i>rindex()</i>
<i>bcopy()</i>	<i>gethostbyname()</i>	<i>scalb()</i>
<i>bsd_signal()</i>	<i>getwd()</i>	<i>setcontext()</i>
<i>bzero()</i>	<i>h_errno</i>	<i>swapcontext()</i>
<i>ecvt()</i>	<i>index()</i>	<i>ualarm()</i>
<i>fcvt()</i>	<i>makecontext()</i>	<i>usleep()</i>
<i>ftime()</i>	<i>mktemp()</i>	<i>vfork()</i>
<i>gcvt()</i>	<i>pthread_attr_getstackaddr()</i>	<i>wcswcs()</i>
<i>getcontext()</i>	<i>pthread_attr_setstackaddr()</i>	

The *utimes()* function, which was marked legacy in Issue 6, has been reinstated as mandatory in Issue 7.

- Functionality associated with the following options in Issue 6 is now mandatory in Issue 7:
 - _POSIX_ASYNCHRONOUS_IO
 - _POSIX_BARRIERS
 - _POSIX_CLOCK_SELECTION
 - _POSIX_MAPPED_FILES
 - _POSIX_MEMORY_PROTECTION
 - _POSIX_READER_WRITER_LOCKS
 - _POSIX_REALTIME_SIGNALS
 - _POSIX_SEMAPHORES
 - _POSIX_SPIN_LOCKS
 - _POSIX_THREAD_SAFE_FUNCTIONS
 - _POSIX_THREADS
 - _POSIX_TIMEOUTS
 - _POSIX_TIMERS

New Features in Issue 7

The functions first introduced in Issue 7 (over the XSH, Issue 6 base document) are listed in the table below:

New Functions in Issue 7		
<i>alphasort()</i>	<i>iswdigit_I()</i>	<i>strfmon_I()</i>
<i>dirfd()</i>	<i>iswgraph_I()</i>	<i>strncasecmp_I()</i>
<i>dprintf()</i>	<i>iswlower_I()</i>	<i>strndup()</i>
<i>duplocale()</i>	<i>iswprint_I()</i>	<i>strlen()</i>
<i>faccessat()</i>	<i>iswpunct_I()</i>	<i>strsignal()</i>
<i>fchmodat()</i>	<i>iswspace_I()</i>	<i>strxfrm_I()</i>
<i>fchownat()</i>	<i>iswupper_I()</i>	<i>symlinkat()</i>
<i>fdopendir()</i>	<i>iswxdigit_I()</i>	<i>tolower_I()</i>
<i>fexecve()</i>	<i>isxdigit_I()</i>	<i>toupper_I()</i>
<i>fmemopen()</i>	<i>linkat()</i>	<i>towctrans_I()</i>
<i>freelocale()</i>	<i>mbsnrtowcs()</i>	<i>towlower_I()</i>
<i>fstatat()</i>	<i>mkdirat()</i>	<i>toupper_I()</i>
<i>futimens()</i>	<i>mkdtemp()</i>	<i>unlinkat()</i>
<i>getdelim()</i>	<i>mkfifoat()</i>	<i>uselocale()</i>
<i>getline()</i>	<i>mknodat()</i>	<i>utimensat()</i>
<i>isalnum_I()</i>	<i>newlocale()</i>	<i>vdprintf()</i>
<i>isalpha_I()</i>	<i>openat()</i>	<i>wcpcpy()</i>
<i>isblank_I()</i>	<i>open_memstream()</i>	<i>wcpncpy()</i>
<i>iscntrl_I()</i>	<i>open_wmemstream()</i>	<i>wscasecmp()</i>
<i>isdigit_I()</i>	<i>psiginfo()</i>	<i>wscasecmp_I()</i>
<i>isgraph_I()</i>	<i>psignal()</i>	<i>wscoll_I()</i>
<i>islower_I()</i>	<i>pthread_mutexattr_getrobust()</i>	<i>wcsdup()</i>
<i>isprint_I()</i>	<i>pthread_mutexattr_setrobust()</i>	<i>wcsncasecmp()</i>
<i>ispunct_I()</i>	<i>pthread_mutex_consistent()</i>	<i>wcsncasecmp_I()</i>
<i>isspace_I()</i>	<i>readlinkat()</i>	<i>wcsnlen()</i>
<i>isupper_I()</i>	<i>renameat()</i>	<i>wcsnrtombs()</i>
<i>iswalnum_I()</i>	<i>scandir()</i>	<i>wcsxfrm_I()</i>
<i>iswalpha_I()</i>	<i>stpncpy()</i>	<i>wctrans_I()</i>
<i>iswblank_I()</i>	<i>stpncpy()</i>	<i>wctype_I()</i>
<i>iswcntrl_I()</i>	<i>strcasecmp_I()</i>	
<i>iswctype_I()</i>	<i>strcoll_I()</i>	

The following table lists functions that were previously part of an option group but are now mandatory in Issue 7:

Newly Mandated Functions in Issue 7		
<i>aio_cancel()</i>	<i>pthread_atfork()</i>	<i>pthread_rwlock_tryrdlock()</i>
<i>aio_error()</i>	<i>pthread_attr_destroy()</i>	<i>pthread_rwlock_trywrlock()</i>
<i>aio_fsync()</i>	<i>pthread_attr_getdetachstate()</i>	<i>pthread_rwlock_unlock()</i>
<i>aio_read()</i>	<i>pthread_attr_getguardsize()</i>	<i>pthread_rwlock_wrlock()</i>
<i>aio_return()</i>	<i>pthread_attr_getschedparam()</i>	<i>pthread_rwlockattr_destroy()</i>
<i>aio_suspend()</i>	<i>pthread_attr_init()</i>	<i>pthread_rwlockattr_init()</i>
<i>aio_write()</i>	<i>pthread_attr_setdetachstate()</i>	<i>pthread_self()</i>
<i>asctime_r()</i>	<i>pthread_attr_setguardsize()</i>	<i>pthread_setcancelstate()</i>
<i>catclose()</i>	<i>pthread_attr_setschedparam()</i>	<i>pthread_setcanceltype()</i>
<i>catgets()</i>	<i>pthread_barrier_destroy()</i>	<i>pthread_setspecific()</i>
<i>catopen()</i>	<i>pthread_barrier_init()</i>	<i>pthread_spin_destroy()</i>
<i>clock_getres()</i>	<i>pthread_barrier_wait()</i>	<i>pthread_spin_init()</i>
<i>clock_gettime()</i>	<i>pthread_barrierattr_destroy()</i>	<i>pthread_spin_lock()</i>
<i>clock_nanosleep()</i>	<i>pthread_barrierattr_init()</i>	<i>pthread_spin_trylock()</i>
<i>clock_settime()</i>	<i>pthread_cancel()</i>	<i>pthread_spin_unlock()</i>
<i>ctime_r()</i>	<i>pthread_cleanup_pop()</i>	<i>pthread_testcancel()</i>
<i>dlclose()</i>	<i>pthread_cleanup_push()</i>	<i>putc_unlocked()</i>
<i>dLError()</i>	<i>pthread_cond_broadcast()</i>	<i>putchar_unlocked()</i>
<i>dlopen()</i>	<i>pthread_cond_destroy()</i>	<i>pwrite()</i>
<i>dlsym()</i>	<i>pthread_cond_init()</i>	<i>rand_r()</i>
<i>fchdir()</i>	<i>pthread_cond_signal()</i>	<i>readdir_r()</i>
<i>flockfile()</i>	<i>pthread_cond_timedwait()</i>	<i>sem_close()</i>
<i>fstatvfs()</i>	<i>pthread_cond_wait()</i>	<i>sem_destroy()</i>
<i>ftrylockfile()</i>	<i>pthread_condattr_destroy()</i>	<i>sem_getvalue()</i>
<i>funlockfile()</i>	<i>pthread_condattr_getclock()</i>	<i>sem_init()</i>
<i>getc_unlocked()</i>	<i>pthread_condattr_init()</i>	<i>sem_open()</i>
<i>getchar_unlocked()</i>	<i>pthread_condattr_setclock()</i>	<i>sem_post()</i>
<i>getgid_r()</i>	<i>pthread_create()</i>	<i>sem_timedwait()</i>
<i>getgnam_r()</i>	<i>pthread_detach()</i>	<i>sem_trywait()</i>
<i>getlogin_r()</i>	<i>pthread_equal()</i>	<i>sem_unlink()</i>
<i>getpgid()</i>	<i>pthread_exit()</i>	<i>sem_wait()</i>
<i>getpwnam_r()</i>	<i>pthread_getspecific()</i>	<i>sigqueue()</i>
<i>getpwuid_r()</i>	<i>pthread_join()</i>	<i>sigqueue()</i>
<i>getsid()</i>	<i>pthread_key_create()</i>	<i>sigtimedwait()</i>
<i>getsubopt()</i>	<i>pthread_key_delete()</i>	<i>sigwaitinfo()</i>
<i>gmtime_r()</i>	<i>pthread_mutex_destroy()</i>	<i>statvfs()</i>
<i>iconv()</i>	<i>pthread_mutex_init()</i>	<i>strcasecmp()</i>
<i>iconv_close()</i>	<i>pthread_mutex_lock()</i>	<i>strdup()</i>
<i>iconv_open()</i>	<i>pthread_mutex_timedlock()</i>	<i>strerror_r()</i>
<i>lchown()</i>	<i>pthread_mutex_trylock()</i>	<i>strfmon()</i>
<i>lio_listio()</i>	<i>pthread_mutex_unlock()</i>	<i>strncasecmp()</i>
<i>localtime_r()</i>	<i>pthread_mutexattr_destroy()</i>	<i>strtok_r()</i>
<i>mkstemp()</i>	<i>pthread_mutexattr_gettype()</i>	<i>tcgetsid()</i>
<i>mmap()</i>	<i>pthread_mutexattr_init()</i>	<i>timer_create()</i>
<i>mprotect()</i>	<i>pthread_mutexattr_settype()</i>	<i>timer_delete()</i>
<i>munmap()</i>	<i>pthread_once()</i>	<i>timer_getoverrun()</i>
<i>nanosleep()</i>	<i>pthread_rwlock_destroy()</i>	<i>timer_gettime()</i>
<i>nl_langinfo()</i>	<i>pthread_rwlock_init()</i>	<i>timer_settime()</i>
<i>poll()</i>	<i>pthread_rwlock_rdlock()</i>	<i>truncate()</i>
<i>posix_trace_timedgetnext_event()</i>	<i>pthread_rwlock_timedrdlock()</i>	<i>ttyname_r()</i>
<i>pread()</i>	<i>pthread_rwlock_timedwrlock()</i>	<i>waitid()</i>

3.3 System Interfaces by Category

The Single UNIX Specification, Version 4 supports and is aligned with key formal standards, of which POSIX.1-2008 forms the core. This section gives lists of interfaces categorized by functional grouping:

Asynchronous Input and Output Interfaces

aio_cancel(), *aio_error()*, *aio_fsync()*, *aio_read()*, *aio_return()*, *aio_suspend()*,
aio_write(), *lio_listio()*

Barrier Interfaces

pthread_barrier_destroy(), *pthread_barrier_init()*, *pthread_barrier_wait()*,
pthread_barrierattr()

Jump Interfaces

longjmp(), *setjmp()*

Maths Library Interfaces

acos(), *acosf()*, *acosh()*, *acoshf()*, *acoshl()*, *acosl()*, *asin()*, *asinf()*, *asinh()*, *asinhf()*,
asinhl(), *asinl()*, *atan()*, *atan2()*, *atan2f()*, *atan2l()*, *atanf()*, *atanh()*, *atanhf()*, *atanhl()*,
atanl(), *cabs()*, *cabsf()*, *cabsl()*, *cacos()*, *cacosf()*, *cacosh()*, *cacoshf()*, *cacoshl()*,
cacosl(), *carg()*, *cargf()*, *cargl()*, *casin()*, *casinf()*, *casinh()*, *casinhf()*, *casinhl()*, *casinl()*,
catan(), *catanf()*, *catanh()*, *catanhf()*, *catanhl()*, *catanl()*, *cbrt()*, *cbrtf()*, *cbrtl()*, *ccos()*,
ccosf(), *ccosh()*, *ccoshf()*, *ccoshl()*, *ccosl()*, *ceil()*, *ceilf()*, *ceill()*, *cexp()*, *cexpf()*, *cexpl()*,
cimag(), *cimagf()*, *cimagl()*, *clog()*, *clogf()*, *clogl()*, *conj()*, *conjf()*, *conjl()*, *copysign()*,
copysignf(), *copysignl()*, *cos()*, *cosf()*, *cosh()*, *coshf()*, *coshl()*, *cosl()*, *cpow()*, *cpowf()*,
cpowl(), *cproj()*, *cprojf()*, *cprojl()*, *creal()*, *crealf()*, *creall()*, *csin()*, *csinf()*, *csinh()*,
csinhf(), *csinhl()*, *csinl()*, *csqrt()*, *csqrtf()*, *csqrtl()*, *ctan()*, *ctanf()*, *ctanh()*, *ctanhf()*,
ctanhl(), *ctanl()*, *erf()*, *erfc()*, *erfcf()*, *erfcl()*, *erff()*, *erfl()*, *exp()*, *exp2()*, *exp2f()*, *exp2l()*,
expf(), *expl()*, *expm1()*, *expm1f()*, *expm1l()*, *fabs()*, *fabsf()*, *fabsl()*, *fdim()*, *fdimf()*,
fdiml(), *floor()*, *floorf()*, *floorl()*, *fma()*, *fmaf()*, *fmal()*, *fmax()*, *fmaxf()*, *fmaxl()*, *fmin()*,
fminf(), *fminl()*, *fmod()*, *fmodf()*, *fmodl()*, *fpclassify()*, *frexp()*, *frexpf()*, *frexpl()*, *hypot()*,
hypotf(), *hypotl()*, *ilogb()*, *ilogbf()*, *ilogbl()*, *isfinite()*, *isgreater()*, *isgreaterequal()*, *isinf()*,
isless(), *islessequal()*, *islessgreater()*, *isnan()*, *isnormal()*, *isunordered()*, *ldexp()*,
ldexpf(), *ldexpl()*, *lgamma()*, *lgammaf()*, *lgammal()*, *llrint()*, *llrintf()*, *llrintl()*, *llround()*,
llroundf(), *llroundl()*, *log()*, *log10()*, *log10f()*, *log10l()*, *log1p()*, *log1pf()*, *log1pl()*, *log2()*,
log2f(), *log2l()*, *logb()*, *logbf()*, *logbl()*, *logf()*, *logl()*, *lrint()*, *lrintf()*, *lrintl()*, *lround()*,
lroundf(), *lroundl()*, *modf()*, *modff()*, *modfl()*, *nan()*, *nanf()*, *nanl()*, *nearbyint()*,
nearbyintf(), *nearbyintl()*, *nextafter()*, *nextafterf()*, *nextafterl()*, *nexttoward()*,
nexttowardf(), *nexttowardl()*, *pow()*, *powf()*, *powl()*, *remainder()*, *remainderf()*,
remainderl(), *remquo()*, *remquof()*, *remquol()*, *rint()*, *rintf()*, *rintl()*, *round()*, *roundf()*,
roundl(), *scalbln()*, *scalblnf()*, *scalblnl()*, *scalbn()*, *scalbnf()*, *scalbnl()*, *signbit()*, *sin()*,
sinf(), *sinh()*, *sinhf()*, *sinhl()*, *sinl()*, *sqrt()*, *sqrtf()*, *sqrtl()*, *tan()*, *tanf()*, *tanh()*, *tanhf()*,
tanhl(), *tanl()*, *tgamma()*, *tgammaf()*, *tgammaL()*, *trunc()*, *truncf()*, *truncl()*

General ISO C Library Interfaces

abs(), *asctime()*, *atof()*, *atoi()*, *atol()*, *atoll()*, *bsearch()*, *calloc()*, *ctime()*, *difftime()*, *div()*,
feclearexcept(), *fegetenv()*, *fegetexceptflag()*, *fegetround()*, *feholdexcept()*,
feraiseexcept(), *fesetenv()*, *fesetexceptflag()*, *fesetround()*, *fetestexcept()*, *feupdateenv()*,
free(), *gmtime()*, *imaxabs()*, *imaxdiv()*, *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*,
isgraph(), *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *labs()*, *ldiv()*,
llabs(), *lldiv()*, *localeconv()*, *localtime()*, *malloc()*, *memchr()*, *memcmp()*, *memcpy()*,
memmove(), *memset()*, *mktime()*, *qsort()*, *rand()*, *realloc()*, *setlocale()*, *snprintf()*,
sprintf(), *srand()*, *sscanf()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*, *strcpy()*, *strcsn()*,
strerror(), *strftime()*, *strlen()*, *strncat()*, *strncmp()*, *strncpy()*, *strpbrk()*, *strchr()*, *strspn()*,
strstr(), *strtod()*, *strtof()*, *strtoimax()*, *strtok()*, *strtol()*, *strtold()*, *strtoll()*, *strtoul()*,

strtoull(), *strtoumax()*, *strxfrm()*, *time()*, *tolower()*, *toupper()*, *tzname*, *tzset()*, *va_arg()*,
va_copy(), *va_end()*, *va_start()*, *vsprintf()*, *vscanf()*

Thread-Safe General ISO C Library Interfaces

asctime_r(), *ctime_r()*, *gmtime_r()*, *localtime_r()*, *rand_r()*, *strerror_r()*, *strtok_r()*

Wide-Character ISO C Library Interfaces

btowc(), *iswalnum()*, *iswalphabeta()*, *iswblank()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*,
iswlower(), *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *mblen()*, *mbrlen()*,
mbtowc(), *mbsinit()*, *mbsrtowcs()*, *mbstowcs()*, *mbtowc()*, *swprintf()*, *swscanf()*,
towctrans(), *tolower()*, *toupper()*, *vswprintf()*, *vswscanf()*, *wcrtomb()*, *wcscat()*,
wcscchr(), *wcscmp()*, *wcscoll()*, *wcscpy()*, *wcscspn()*, *wcsftime()*, *wcslen()*, *wcsncat()*,
wcsncmp(), *wcsncpy()*, *wcspbrk()*, *wcsrchr()*, *wcsrtombs()*, *wcsspn()*, *wcsstr()*, *wcstod()*,
wcstof(), *wcstoimax()*, *wcstok()*, *wcstol()*, *wcstold()*, *wcstoll()*, *wcstombs()*, *wcstoul()*,
wcstoull(), *wcstoumax()*, *wcsxfrm()*, *wctob()*, *wctomb()*, *wctrans()*, *wctype()*, *wmemchr()*,
wmemcmp(), *wmemcpy()*, *wmemmove()*, *wmemset()*

Extended Wide-Character ISO C Library Interfaces

mbsnrtowcs(), *wcpcpy()*, *wcpncpy()*, *wcscasecmp()*, *wcsdup()*, *wcsncasecmp()*,
wcsnlen(), *wcsnrtombs()*

General C Library Extension Interfaces

fnmatch(), *getopt()*, *getsubopt()*, *optarg*, *opterr*, *optind*, *optopt*, *stpncpy()*, *stpncpy()*,
strcasecmp(), *strdup()*, *strfmon()*, *strncasecmp()*, *strndup()*, *strnlen()*

Clock Selection Interfaces

clock_nanosleep(), *pthread_condattr_getclock()*, *pthread_condattr_setclock()*

Device Input and Output Interfaces

FD_CLR(), *FD_ISSET()*, *FD_SET()*, *FD_ZERO()*, *clearerr()*, *close()*, *fclose()*, *fdopen()*,
feof(), *ferror()*, *fflush()*, *fgetc()*, *fgets()*, *fileno()*, *fopen()*, *fprintf()*, *fputc()*, *fputs()*, *fread()*,
freopen(), *fscanf()*, *fwrite()*, *getc()*, *getchar()*, *gets()*, *open()*, *perror()*, *poll()*, *printf()*,
pread(), *pselect()*, *putc()*, *putchar()*, *puts()*, *pwrite()*, *read()*, *scanf()*, *select()*, *setbuf()*,
setvbuf(), *stderr*, *stdin*, *stdout*, *ungetc()*, *vfprintf()*, *vfscanf()*, *vprintf()*, *vscanf()*, *write()*

Extended Device Input and Output Interfaces

dprintf(), *fmemopen()*, *open_memstream()*, *vdprintf()*

General Terminal Interfaces

cfgetispeed(), *cfgetospeed()*, *cfsetispeed()*, *cfsetospeed()*, *ctermid()*, *isatty()*, *tcdrain()*,
tcflow(), *tcflush()*, *tcgetattr()*, *tcsendbreak()*, *tcsetattr()*, *ttyname()*

Thread-Safe General Terminal Interfaces

ttyname_r()

Dynamic Linking Interfaces

dldclose(), *dlderror()*, *dlopen()*, *dlsym()*

File Descriptor Management Interfaces

dup(), *dup2()*, *fcntl()*, *fgetpos()*, *fseek()*, *fseeko()*, *fsetpos()*, *ftell()*, *ftello()*, *ftruncate()*,
lseek(), *rewind()*

FIFO Interfaces

mkfifo()

FIFO File Descriptor Interfaces

mkfifoat(), *mknodat()*

File Attribute Interfaces

chmod(), *chown()*, *fchmod()*, *fchown()*, *umask()*

File Attributes File Descriptor Interfaces

fchmodat(), *fchownat()*

Thread-Safe Stdio Locking Interfaces

flockfile(), *ftrylockfile()*, *funlockfile()*, *getc_unlocked()*, *getchar_unlocked()*,
putc_unlocked(), *putchar_unlocked()*

File System Interfaces

access(), *chdir()*, *closedir()*, *creat()*, *fchdir()*, *fpathconf()*, *fstat()*, *fstatvfs()*, *getcwd()*,
link(), *mkdir()*, *mkstemp()*, *opendir()*, *pathconf()*, *readdir()*, *remove()*, *rename()*,
rewinddir(), *rmdir()*, *stat()*, *statvfs()*, *tmpfile()*, *tmpnam()*, *truncate()*, *unlink()*, *utime()*

File System Extension Interfaces

alphasort(), *dirfd()*, *getdelim()*, *getline()*, *mkdtemp()*, *scandir()*

File System File Descriptor Interfaces

faccessat(), *fdopendir()*, *fstatat()*, *linkat()*, *mkdirat()*, *openat()*, *renameat()*, *unlinkat()*,
utimensat()

File System Glob Expansion Interfaces

glob(), *globfree()*

Thread-Safe File System Interfaces

readdir_r()

Internationalization Interfaces

catclose(), *catgets()*, *catopen()*, *iconv()*, *iconv_close()*, *iconv_open()*, *nl_langinfo()*

Job Control Interfaces

setpgid(), *tcgetpgrp()*, *tcsetpgrp()*, *tcgetsid()*

Memory Mapped File Interfaces

mmap(), *munmap()*

Memory Protection Interfaces

mprotect()

Multiple Concurrent Locale Interfaces

duplocale(), *freelocale()*, *isalnum_I()*, *isalpha_I()*, *isblank_I()*, *iscntrl_I()*, *isdigit_I()*,
isgraph_I(), *islower_I()*, *isprint_I()*, *ispunct_I()*, *isspace_I()*, *isupper_I()*, *iswalnum_I()*,
iswalpha_I(), *iswblank_I()*, *iswcntrl_I()*, *iswctype_I()*, *iswdigit_I()*, *iswgraph_I()*,
iswlower_I(), *iswprint_I()*, *iswpunct_I()*, *iswspace_I()*, *iswupper_I()*, *iswxdigit_I()*,
isxdigit_I(), *newlocale()*, *strcasecmp_I()*, *strcoll_I()*, *strfmon_I()*, *strncasecmp_I()*,
strxfrm_I(), *tolower_I()*, *toupper_I()*, *towctrans_I()*, *tolower()*, *toupper()*, *uselocale()*,
wscasecmp_I(), *wscoll_I()*, *wcsncasecmp_I()*, *wcsxfrm_I()*, *wctrans_I()*, *wctype_I()*

Multiple Process Interfaces

_Exit(), *_exit()*, *assert()*, *atexit()*, *clock()*, *execl()*, *execle()*, *execlp()*, *execv()*, *execve()*,
execvp(), *exit()*, *fork()*, *getpgrp()*, *getpgid()*, *getpid()*, *getppid()*, *getsid()*, *setsid()*,
sleep(), *times()*, *wait()*, *waitid()*, *waitpid()*

Multiple Processes File Descriptor Interfaces

fexecve()

Networking Interfaces

accept(), *bind()*, *connect()*, *endhostent()*, *endnetent()*, *endprotoent()*, *endservent()*,
freeaddrinfo(), *gai_strerror()*, *getaddrinfo()*, *gethostent()*, *gethostname()*, *getnameinfo()*,

getnetbyaddr(), getnetbyname(), getnetent(), getpeername(), getprotobyname(), getprotobyname(), getprotoent(), getservbyname(), getservbyport(), getservent(), getsockname(), getsockopt(), htonl(), htons(), if_freenameindex(), if_indexname(), if_nameindex(), if_nametoindex(), inet_addr(), inet_ntoa(), inet_ntop(), inet_pton(), listen(), ntohl(), ntohs(), recv(), recvfrom(), recvmsg(), send(), sendmsg(), sendto(), sethostent(), setnetent(), setprotoent(), setservent(), setsockopt(), shutdown(), socket(), socketat(), socketpair()

Pipe Interfaces

pipe()

Robust Mutex Interfaces

pthread_mutex_consistent(), pthread_mutexattr_getrobust(), pthread_mutexattr_setrobust()

Realtime Signal Interfaces

sigqueue(), sigtimedwait(), sigwaitinfo()

Regular Expression Interfaces

regcomp(), regerror(), regexec(), regfree()

Reader Writer Lock Interfaces

pthread_rwlock_destroy(), pthread_rwlock_init(), pthread_rwlock_rdlock(), pthread_rwlock_timedrdlock(), pthread_rwlock_timedwrlock(), pthread_rwlock_tryrdlock(), pthread_rwlock_trywrlock(), pthread_rwlock_unlock(), pthread_rwlock_wrlock(), pthread_rwlockattr_destroy(), pthread_rwlockattr_init(), pthread_rwlockattr_getpshared(), pthread_rwlockattr_setpshared()

Semaphore Interfaces

sem_close(), sem_destroy(), sem_getvalue(), sem_init(), sem_open(), sem_post(), sem_timedwait(), sem_trywait(), sem_unlink(), sem_wait()

Shell and Utilities Interfaces

pclose(), popen(), system(), wordexp(), wordfree()

Signal Jump Interfaces

siglongjmp(), sigsetjmp()

Signal Interfaces

abort(), alarm(), kill(), pause(), raise(), sigaction(), sigaddset(), sigdelset(), sigemptyset(), sigfillset(), sigismember(), signal(), sigpending(), sigprocmask(), sigsuspend(), sigwait()

Extended Signal Interfaces

psignal(), psigno(), strsignal()

Single Process Interfaces

confstr(), environ(), errno(), getenv(), setenv(), sysconf(), uname(), unsetenv()

Spin Lock Interfaces

pthread_spin_destroy(), pthread_spin_init(), pthread_spin_lock(), pthread_spin_trylock(), pthread_spin_unlock()

Symbolic Link Interfaces

lchown(), lstat(), readlink(), symlink()

Symbolic Link File Descriptor Interfaces

readlinkat(), symlinkat()

System Database Interfaces

getrgid(), getgrnam(), getpwnam(), getpwuid()

Thread-Safe System Database Interfaces

getrgid_r(), getgrnam_r(), getpwnam_r(), getpwuid_r()

Base Threads Interfaces

pthread_atfork(), pthread_attr_destroy(), pthread_attr_getdetachstate(), pthread_attr_getschedparam(), pthread_attr_init(), pthread_attr_setdetachstate(), pthread_attr_setschedparam(), pthread_cancel(), pthread_cleanup_pop(), pthread_cleanup_push(), pthread_cond_broadcast(), pthread_cond_destroy(), pthread_cond_init(), pthread_cond_signal(), pthread_cond_timedwait(), pthread_cond_wait(), pthread_condattr_destroy(), pthread_condattr_init(), pthread_create(), pthread_detach(), pthread_equal(), pthread_exit(), pthread_getspecific(), pthread_join(), pthread_key_create(), pthread_key_delete(), pthread_kill(), pthread_mutex_destroy(), pthread_mutex_init(), pthread_mutex_lock(), pthread_mutex_timedlock(), pthread_mutex_trylock(), pthread_mutex_unlock(), pthread_mutexattr_destroy(), pthread_mutexattr_init(), pthread_once(), pthread_self(), pthread_setcancelstate(), pthread_setcanceltype(), pthread_setspecific(), pthread_sigmask(), pthread_testcancel()

Extended Threads Interfaces

pthread_attr_getguardsize(), pthread_attr_setguardsize(), pthread_mutexattr_gettype(), pthread_mutexattr_settype()

Timer Interfaces

clock_getres(), clock_gettime(), clock_settime(), nanosleep(), timer_create(), timer_delete(), timer_getoverrun(), timer_gettime(), timer_settime()

User and Group Interfaces

getegid(), geteuid(), getgid(), getgroups(), getlogin(), getuid(), setegid(), seteuid(), setgid(), setuid()

Thread-Safe User and Group Interfaces

getlogin_r()

Wide Character Device Input and Output Interfaces

fgetwc(), fgetws(), fputwc(), fputws(), fwide(), fwprintf(), fwscanf(), getwc(), getwchar(), putwc(), putwchar(), ungetwc(), vfwprintf(), vfwscanf(), vwprintf(), vwscanf(), wprintf(), wscanf()

Advisory Information Interfaces

posix_fadvise(), posix_fallocate(), posix_madvise(), posix_memalign()

Process CPU-Time Clocks Interfaces

clock_getcpuclockid()

File Synchronization Interfaces

fsync()

Process Memory Locking Interfaces

mlockall(), munlockall()

Range Memory Locking Interfaces

mlock(), munlock()

Message Passing Interfaces

mq_close(), mq_getattr(), mq_notify(), mq_open(), mq_receive(), mq_send(), mq_setattr(), mq_timedreceive(), mq_timedsend(), mq_unlink()

Process Scheduling Interfaces

posix_spawnattr_getschedparam(), *posix_spawnattr_getschedpolicy()*,
posix_spawnattr_setschedparam(), *posix_spawnattr_setschedpolicy()*,
sched_get_priority_max(), *sched_get_priority_min()*, *sched_getparam()*,
sched_getscheduler(), *sched_rr_get_interval()*, *sched_setparam()*, *sched_setscheduler()*

Robust Mutex Priority Inheritance Interfaces

pthread_mutexattr_getprotocol(), *pthread_mutexattr_setprotocol()*

Robust Mutex Priority Protection Interfaces

pthread_mutex_getprioceiling(), *pthread_mutex_setprioceiling()*,
pthread_mutexattr_getprioceiling(), *pthread_mutexattr_getprotocol()*,
pthread_mutexattr_setprioceiling(), *pthread_mutexattr_setprotocol()*

Shared Memory Objects Interfaces

shm_open(), *shm_unlink()*

Synchronized Input and Output Interfaces

fdatasync(), *msync()*

Spawn Interfaces

posix_spawn(), *posix_spawn_file_actions_addclose()*,
posix_spawn_file_actions_adddup2(), *posix_spawn_file_actions_addopen()*,
posix_spawn_file_actions_destroy(), *posix_spawn_file_actions_init()*,
posix_spawnattr_destroy(), *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*,
posix_spawnattr_getschedparam(), *posix_spawnattr_getschedpolicy()*,
posix_spawnattr_getsigdefault(), *posix_spawnattr_getsigmask()*, *posix_spawnattr_init()*,
posix_spawnattr_setflags(), *posix_spawnattr_setpgroup()*,
posix_spawnattr_setschedparam(), *posix_spawnattr_setschedpolicy()*,
posix_spawnattr_setsigdefault(), *posix_spawnattr_setsigmask()*, *posix_spawnnp()*

Thread CPU-Time Clocks Interfaces

pthread_getcpuclockid()

Trace Event Filter Interfaces

posix_trace_eventset_add(), *posix_trace_eventset_del()*, *posix_trace_eventset_empty()*,
posix_trace_eventset_fill(), *posix_trace_eventset_ismember()*, *posix_trace_get_filter()*,
posix_trace_set_filter(), *posix_trace_trid_eventid_open()*

Non-Robust Mutex Priority Inheritance Interfaces

pthread_mutexattr_getprotocol(), *pthread_mutexattr_setprotocol()*

Non-Robust Mutex Priority Protection Interfaces

pthread_mutex_getprioceiling(), *pthread_mutex_setprioceiling()*,
pthread_mutexattr_getprioceiling(), *pthread_mutexattr_getprotocol()*,
pthread_mutexattr_setprioceiling(), *pthread_mutexattr_setprotocol()*

Thread Execution Scheduling Interfaces

pthread_attr_getinheritsched(), *pthread_attr_getschedpolicy()*, *pthread_attr_getscope()*,
pthread_attr_setinheritsched(), *pthread_attr_setschedpolicy()*, *pthread_attr_setscope()*,
pthread_getschedparam(), *pthread_setschedparam()*, *pthread_setschedprio()*,
sched_get_priority_max(), *sched_get_priority_min()*, *sched_rr_get_interval()*

Trace Interfaces

posix_trace_attr_destroy(), *posix_trace_attr_getclockres()*,
posix_trace_attr_getcreatetime(), *posix_trace_attr_getgenversion()*,
posix_trace_attr_getname(), *posix_trace_attr_getstreamfullpolicy()*,
posix_trace_attr_getstreamsize(), *posix_trace_attr_init()*,

posix_trace_attr_setmaxdatasize(), *posix_trace_attr_setname()*,
posix_trace_attr_setstreamfullpolicy(), *posix_trace_attr_setstreamsize()*,
posix_trace_clear(), *posix_trace_create()*, *posix_trace_event()*,
posix_trace_eventid_equal(), *posix_trace_eventid_get_name()*,
posix_trace_eventid_open(), *posix_trace_eventtypelist_getnext_id()*,
posix_trace_eventtypelist_rewind(), *posix_trace_get_attr()*, *posix_trace_get_status()*,
posix_trace_getnext_event(), *posix_trace_shutdown()*, *posix_trace_start()*,
posix_trace_stop(), *posix_trace_timedgetnext_event()*, *posix_trace_trygetnext_event()*

Trace Inherit Interfaces

posix_trace_attr_getinherited(), *posix_trace_attr_setinherited()*

Trace Log Interfaces

posix_trace_attr_getlogfullpolicy(), *posix_trace_attr_getlogsize()*,
posix_trace_attr_setlogfullpolicy(), *posix_trace_attr_setlogsize()*, *posix_trace_close()*,
posix_trace_create_withlog(), *posix_trace_flush()*, *posix_trace_open()*,
posix_trace_rewind()

Thread Stack Address Attribute Interfaces

pthread_attr_getstack(), *pthread_attr_setstack()*

Thread Process-Shared Synchronization Interfaces

pthread_barrierattr_getpshared(), *pthread_barrierattr_setpshared()*,
pthread_condattr_getpshared(), *pthread_condattr_setpshared()*,
pthread_mutexattr_getpshared(), *pthread_mutexattr_setpshared()*,
pthread_rwlockattr_getpshared(), *pthread_rwlockattr_setpshared()*

Thread Stack Size Attribute Interfaces

pthread_attr_getstack(), *pthread_attr_getstacksize()*, *pthread_attr_setstack()*,
pthread_attr_setstacksize()

Typed Memory Objects Interfaces

posix_mem_offset(), *posix_typed_mem_get_info()*, *posix_typed_mem_open()*

XSI General C Library Interfaces

_tolower(), *_toupper()*, *a64l()*, *daylight()*, *drand48()*, *erand48()*, *ffs()*, *getdate()*,
hcreate(), *hdestroy()*, *hsearch()*, *initstate()*, *insque()*, *isascii()*, *jrand48()*, *l64a()*,
lcong48(), *lfind()*, *lrand48()*, *lsearch()*, *memccpy()*, *mrand48()*, *nrand48()*, *random()*,
remque(), *seed48()*, *setstate()*, *signgam*, *srand48()*, *srandom()*, *strptime()*, *swab()*,
tdelete(), *tfind()*, *timezone()*, *toascii()*, *tsearch()*, *twalk()*

XSI Database Management Interfaces

dbm_clearerr(), *dbm_close()*, *dbm_delete()*, *dbm_error()*, *dbm_fetch()*, *dbm_firstkey()*,
dbm_nextkey(), *dbm_open()*, *dbm_store()*

XSI Device Input and Output Interfaces

fmtmsg(), *readv()*, *writev()*

XSI General Terminal Interfaces

grantpt(), *posix_openpt()*, *ptsname()*, *unlockpt()*

XSI File System Interfaces

basename(), *dirname()*, *ftw()*, *lockf()*, *mknod()*, *nftw()*, *realpath()*, *seekdir()*, *sync()*,
telldir(), *tempnam()*

XSI Interprocess Communication Interfaces

ftok(), *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, *semctl()*, *semget()*, *semop()*, *shmat()*,
shmctl(), *shmdt()*, *shmget()*

XSI Jump Interfaces

_longjmp(), *_setjmp()*

XSI Maths Library Interfaces

j0(), *j1()*, *jn()*, *y0()*, *y1()*, *yn()*

XSI Multiple Process Interfaces

getpriority(), *getrlimit()*, *getrusage()*, *nice()*, *setpgrp()*, *setpriority()*, *setrlimit()*, *ulimit()*,

XSI Signal Interfaces

killpg(), *sigaltstack()*, *sighold()*, *sigignore()*, *siginterrupt()*, *sigpause()*, *sigrelse()*, *sigset()*,

XSI Single Process Interfaces

gethostid(), *gettimeofday()*, *putenv()*

XSI System Database Interfaces

endpwent(), *getpwent()*, *setpwent()*

XSI System Logging Interfaces

closelog(), *openlog()*, *setlogmask()*, *syslog()*

XSI Threads Extension Interfaces

pthread_attr_getstack(), *pthread_attr_setstack()*, *pthread_getconcurrency()*,
pthread_setconcurrency()

XSI Timer Interfaces

getitimer(), *setitimer()*

XSI User and Group Interfaces

endgrent(), *endutxent()*, *getgrent()*, *getutxent()*, *getutxid()*, *getutxline()*, *pututxline()*,
setgrent(), *setregid()*, *setreuid()*, *setutxent()*

XSI Wide-Character Library Interfaces

wcswidth(), *wcwidth()*

XSI STREAMS Interfaces

fattach(), *fdetach()*, *getmsg()*, *getpmsg()*, *ioctl()*, *isastream()*, *putmsg()*, *putpmsg()*

3.4 XSH Option Groups

The Single UNIX Specification includes a set of profiling options, allowing larger profiles of the options of the Base standard. In Version 2 and earlier versions of the Single UNIX Specification these were known as Feature Groups. The Option Groups within the Single UNIX Specification are defined within XBD, Section 2.1.5.2, XSI Option Groups.

The Single UNIX Specification contains the following Option Groups relating to XSH:

- **Encryption**, covering the functions *crypt()*, *encrypt()*, and *setkey()*
- **Realtime**, covering the functions from the IEEE Std 1003.1b-1993 Realtime extension

This Option Group consists of the set of the following options from within POSIX.1-2008:


```

_POSIX_FSYNC
_POSIX_MEMLOCK
_POSIX_MEMLOCK_RANGE
_POSIX_MESSAGE_PASSING
_POSIX_PRIORITIZED_IO
_POSIX_PRIORITY_SCHEDULING
_POSIX_SHARED_MEMORY_OBJECTS
_POSIX_SYNCHRONIZED_IO

```

Where entire functions are included in the Option Group, the NAME section is marked with REALTIME. Where additional semantics have been added to existing pages, the new material is identified by use of the appropriate margin legend for the underlying option defined within POSIX.1-2008.

- **Realtime Threads**, covering the functions from the IEEE Std 1003.1c-1995 Threads extension that are related to realtime functionality

This Option Group consists of the set of the following options from within POSIX.1-2008:

```

_POSIX_THREAD_PRIO_INHERIT
_POSIX_THREAD_PRIO_PROTECT
_POSIX_THREAD_PRIORITY_SCHEDULING

```

Where entire functions are included in the Option Group, the NAME section is marked with REALTIME THREADS. Where additional semantics have been added to existing pages, the new material is identified by use of the appropriate margin legend for the underlying option defined within POSIX.1-2008.

- **Advanced Realtime**, covering some of the non-threads-related functions from IEEE Std 1003.1d-1999 and IEEE Std 1003.1j-2000

This Option Group consists of the set of the following options from within POSIX.1-2008:

```

_POSIX_ADVISORY_INFO
_POSIX_CPUTIME
_POSIX_MONOTONIC_CLOCK
_POSIX_SPAWN
_POSIX_SPORADIC_SERVER
_POSIX_TYPED_MEMORY_OBJECTS

```

Where entire functions are included in the Option Group, the NAME section is marked with ADVANCED REALTIME. Where additional semantics have been added to existing pages, the new material is identified by use of the appropriate margin legend for the underlying option defined within POSIX.1-2008.

- **Advanced Realtime Threads**, covering some of the threads-related functions from IEEE Std 1003.1d-1999 and IEEE Std 1003.1j-2000

This Option Group consists of the set of the following options from within POSIX.1-2008:

```

_POSIX_THREAD_CPUTIME
_POSIX_THREAD_SPORADIC_SERVER

```

Where entire functions are included in the Option Group, the NAME section is marked with ADVANCED REALTIME THREADS. Where additional semantics have been added to existing pages, the new material is identified by use of the appropriate margin legend for the underlying option defined within POSIX.1-2008.

- **Tracing**, covering the functionality from IEEE Std 1003.1q-2000

This Option Group consists of the set of the following options from within POSIX.1-2008:

```
_POSIX_TRACE
_POSIX_TRACE_EVENT_FILTER
_POSIX_TRACE_LOG
_POSIX_TRACE_INHERIT
```

Where entire functions are included in the Option Group, the NAME section is marked with TRACING. Where additional semantics have been added to existing pages, the new material is identified by use of the appropriate margin legend for the underlying option defined within POSIX.1-2008.

- **XSI STREAMS**, covering the functionality and interfaces related to STREAMS, a uniform mechanism for implementing networking services and other character-based I/O as described in XSH, Section 2.6, STREAMS

This option group includes the following functions:

```
fattach(), fdetach(), getmsg(), getpmsg(), ioctl(), isastream(), putmsg(), putpmsg()
```

and the **<stropts.h>** header.

Where applicable, whole pages are marked STREAMS, together with the appropriate option margin legend for the SYNOPSIS section. Where additional semantics have been added to existing pages, the new material is identified by use of the appropriate margin legend for the underlying option defined within POSIX.1-2008.

3.5 Options Policy

In Version 2 and earlier versions of the Single UNIX Specification, conforming systems had to provide headers, and stub functions returning the [ENOSYS] error condition for unsupported options.

The policy introduced in Version 3 and updated in this version of the specification provides flexibility for implementations in how they support options. It also specifies how conforming applications can adapt to different implementations that support different sets of options. It allows the following:

1. If an implementation has no interest in supporting an option, it does not have to provide anything associated with that option beyond the announcement that it does not support it.
2. An implementation can support a partial or incompatible version of an option (as a non-standard extension) as long as it does not claim to support the option.
3. An application can determine whether the option is supported. A strictly conforming application must check this announcement mechanism before first using anything associated with the option.

There is an important implication of this policy. XSH does not dictate the behavior of interfaces associated with an option when the implementation does not claim to support the option. In particular, it does not require that a function associated with an unsupported option will fail if it does not perform as specified.

The policy is enabled through the symbolic constants for Options and Option Groups defined in **<unistd.h>**.

Any application must consider the following three cases for each option:

1. Option not supported for compilation

The implementation advertises at compile time (by defining the constant in **<unistd.h>** with value `-1`, or by leaving it undefined) that the option is not supported for compilation and, at the time of compilation, is not supported for runtime use. In this case, the headers, data types, function interfaces, and utilities required only for the option need not be present. A later runtime check using the `fpathconf()`, `pathconf()`, or `sysconf()` functions defined in the System Interfaces volume of POSIX.1-2008 or the `getconf` utility defined in the Shell and Utilities volume of POSIX.1-2008 can in some circumstances indicate that the option is supported at runtime. (For example, an old application binary might be run on a newer implementation to which support for the option has been added.)

2. Option always supported

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

3. Option might or might not be supported at runtime

The implementation advertises at compile time (by defining the constant in **<unistd.h>** with value zero) that the option is supported for compilation and might or might not be supported at runtime. In this case, the `fpathconf()`, `pathconf()`, or `sysconf()` functions defined in the System Interfaces volume of POSIX.1-2008 or the `getconf` utility defined in the Shell and Utilities volume of POSIX.1-2008 can be used to retrieve the value of each symbol on each specific implementation to determine whether the option is supported at runtime. All headers, data types, and function interfaces required to compile and execute applications which use the option at runtime (after checking at runtime that the option is supported) shall be provided, but if the option is not supported at runtime they need not operate as specified. Utilities or other facilities required only for the option, but not needed to compile and execute such applications, need not be present.

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

Shell and Utilities

XCU, Issue 7 describes all of the utilities required in the Single UNIX Specification environment. This chapter covers the contents of XCU, describing options, functional overview, the scope of the interfaces, an overview of the changes in this version, and lists of the new features. It also contains rationale of why certain utilities are excluded.

4.1 Options in XCU

Some of the utilities need not be present on a system conforming to the Single UNIX Specification. The optional utilities are as follows:

- DEVELOPMENT utilities are those required in a software development environment. The reference pages for these utilities are marked DEVELOPMENT.
- FORTRAN utilities are required in a FORTRAN-77 development environment, and essentially consist of the compiler, *fort77*, and the *ctags* utility. The reference pages for these utilities are marked FORTRAN. (The *ctags* utility is also marked DEVELOPMENT.)
- The UUCP Utilities are optional. These are marked by the UU notation in their SYNOPSIS.
- The Batch Environment Services and Utilities are optional. These are marked by the BE notation in their SYNOPSIS. They are also marked OB to indicate that they are obsolescent.

4.2 Functional Overview

The Single UNIX Specification supports a robust tool environment of 174 utilities, described in XCU. This is 160 external utilities and 14 required built-in utilities. The Single UNIX Specification utilities have been quite stable since XPG4 Base, and provide a standards-based tool environment based on UNIX systems practice.

The XSI extensions over the base POSIX functionality in XCU include utilities such as:

- The SCCS suite of tools for source code management:
admin, delta, get, prs, rmdel, sact, sccs, unget, val, what
- The optional UUCP Utilities for UNIX-to-UNIX file copying:
uucp, uustat, uux

- The compression utilities:
compress, uncompress, zcat
- Additional C-language development tools:
cflow, cxref
- Additional utilities:
cal, fuser, ipcrm, ipcs, link, unlink
- Other miscellaneous tools and utilities. (These tools can be found in the table in [Chapter 8](#) (on page 81), with an **xsi** code in the POSIX column.)

The Single UNIX Specification utilities include:

<i>admin</i>	<i>df</i>	<i>lex</i>	<i>pwd</i>	<i>time</i>
<i>alias</i>	<i>diff</i>	<i>link</i>	<i>qalter</i>	<i>touch</i>
<i>ar</i>	<i>dirname</i>	<i>ln</i>	<i>qdel</i>	<i>tput</i>
<i>asa</i>	<i>du</i>	<i>locale</i>	<i>qhold</i>	<i>tr</i>
<i>at</i>	<i>echo</i>	<i>localedef</i>	<i>qmove</i>	<i>true</i>
<i>awk</i>	<i>ed</i>	<i>logger</i>	<i>qmsg</i>	<i>tsort</i>
<i>basename</i>	<i>env</i>	<i>logname</i>	<i>qrerun</i>	<i>tty</i>
<i>batch</i>	<i>ex</i>	<i>lp</i>	<i>qrls</i>	<i>type</i>
<i>bc</i>	<i>expand</i>	<i>ls</i>	<i>qselect</i>	<i>ulimit</i>
<i>bg</i>	<i>expr</i>	<i>m4</i>	<i>qsig</i>	<i>umask</i>
<i>c99</i>	<i>false</i>	<i>mailx</i>	<i>qstat</i>	<i>unalias</i>
<i>cal</i>	<i>fc</i>	<i>make</i>	<i>qsub</i>	<i>uname</i>
<i>cat</i>	<i>fg</i>	<i>man</i>	<i>read</i>	<i>uncompress</i>
<i>cd</i>	<i>file</i>	<i>mesg</i>	<i>renice</i>	<i>unexpand</i>
<i>cflow</i>	<i>find</i>	<i>mkdir</i>	<i>rm</i>	<i>unget</i>
<i>chgrp</i>	<i>fold</i>	<i>mkfifo</i>	<i>rmdel</i>	<i>uniq</i>
<i>chmod</i>	<i>fort77</i>	<i>more</i>	<i>rmdir</i>	<i>unlink</i>
<i>chown</i>	<i>fuser</i>	<i>mv</i>	<i>sact</i>	<i>uucp</i>
<i>cksum</i>	<i>genscat</i>	<i>newgrp</i>	<i>sccs</i>	<i>uudecode</i>
<i>cmp</i>	<i>get</i>	<i>nice</i>	<i>sed</i>	<i>uuencode</i>
<i>comm</i>	<i>getconf</i>	<i>nl</i>	<i>sh</i>	<i>uustat</i>
<i>command</i>	<i>getopts</i>	<i>nm</i>	<i>sleep</i>	<i>uux</i>
<i>compress</i>	<i>grep</i>	<i>nohup</i>	<i>sort</i>	<i>val</i>
<i>cp</i>	<i>hash</i>	<i>od</i>	<i>split</i>	<i>vi</i>
<i>crontab</i>	<i>head</i>	<i>paste</i>	<i>strings</i>	<i>wait</i>
<i>csplit</i>	<i>iconv</i>	<i>patch</i>	<i>strip</i>	<i>wc</i>
<i>ctags</i>	<i>id</i>	<i>pathchk</i>	<i>stty</i>	<i>what</i>
<i>cut</i>	<i>ipcrm</i>	<i>pax</i>	<i>tabs</i>	<i>who</i>
<i>cxref</i>	<i>ipcs</i>	<i>pr</i>	<i>tail</i>	<i>write</i>
<i>date</i>	<i>jobs</i>	<i>printf</i>	<i>talk</i>	<i>xargs</i>
<i>dd</i>	<i>join</i>	<i>pr</i>	<i>tee</i>	<i>yacc</i>
<i>delta</i>	<i>kill</i>	<i>ps</i>	<i>test</i>	<i>zcat</i>

A certified UNIX system will provide all the tools in XCU, with the following provisions:

- The DEVELOPMENT utilities need not be provided. These consist of:
admin, cflow, ctags, cxref, delta, get, lex, make, nm, prs, rmdel, sact, sccs, strip, unget, val, what, yacc

- The FORTRAN development utilities need not be provided. These consist of the compiler, *fort77*, and the *ctags* utility (which is also a DEVELOPMENT utility).
- The UUCP Utilities need not be provided. These consist of:

uucp, uustat, uux

- The obsolescent Batch Environment Services and Utilities need not be provided. These consist of:

qalter, qdel, qhold, qmove, qmsg, qrerun, qrls, qselect, qsig, qstat, qsub

If the implementation claims to provide the DEVELOPMENT option, then all the tools in the group must be provided.

It should be noted that the C compiler, *c99*, is not considered part of the DEVELOPMENT group. All certified UNIX systems must provide a way of compiling C-language programs.

4.3 Base Documents

The Shell and Utilities provided in XCU, Issue 7 are drawn from the following base documents:

- IEEE Std 1003.1, 2004 Edition (POSIX-1) (incorporating IEEE Std 1003.1-2001, IEEE Std 1003.1-2001/Cor 1-2002, and IEEE Std 1003.1-2001/Cor 2-2004)
- The Open Group Technical Standard, 2006, Extended API Set Part 1
- ISO/IEC 9899:1999, Programming Languages — C (including ISO/IEC 9899:1999/Cor.1:2001(E), ISO/IEC 9899:1999/Cor.2:2004(E), and ISO/IEC 9899:1999/Cor.3(E))

4.4 Overview of Changes

The following sections give a high-level overview of changes made to XCU since Issue 6. [Chapter 12](#) (on page 299) details the technical changes that have been made to the utilities from Issue 6.

Changes from Issue 6 to Issue 7

The following list summarizes the major changes that were made in XCU from Issue 6 to Issue 7:

- Austin Group defect reports, IEEE Interpretations against IEEE Std 1003.1, and responses to ISO/IEC defect reports against ISO/IEC 9945 are applied.
- The Open Group corrigenda and resolutions are applied.
- Features, marked legacy or obsolescent in the base documents, have been considered for removal in this version. The only such feature in XCU is the obsolescent *-r* option of the *cp* utility, which has been removed.
- A review of the use of fixed path filenames within the standard has been undertaken; for example, the *at*, *batch*, and *crontab* utilities previously had a requirement for use of the directory */usr/lib/cron*.

- The options within the standard have been revised.
 - The Batch Environment Services and Utilities option is marked obsolescent.
 - The UUCP utilities option is added.
 - The User Portability Utilities option is revised so that only the *bg*, *ex*, *fc*, *fg*, *jobs*, *more*, *talk*, and *vi* utilities are included, the rest being moved to the Base.

New Features in Issue 7

There are no new utilities in Issue 7. However, new options have been added to some utilities, and a new primary has been added to the *find* utility, as listed in the table below:

Utility	New Options or Primary in Issue 7
<i>diff</i>	-u -U
<i>find</i>	-path
<i>ln</i>	-L -P
<i>ls</i>	-A -S -k
<i>patch</i>	-u
<i>pathchk</i>	-P
<i>sort</i>	-C
<i>touch</i>	-d

4.5 Exclusion of Utilities

The set of utilities contained in the Single UNIX Specification is drawn from the POSIX.2-1992 standard and XCU, Issue 5, with one change: the substitution of the *c99* utility for the *c89* utility. This section contains information on some of the deliberations that led to this set of utilities, and why certain utilities were excluded.

Many utilities were evaluated by the standard developers; more historical utilities were excluded from the standards than included. The following list contains many common UNIX system utilities that were not included.

This section is limited to a discussion of only those utilities actively or indirectly evaluated by the standard developers, rather than the list of all known UNIX utilities from all its variants.

- adb* The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool. Furthermore, many useful aspects of *adb* are very hardware-specific.
- as* Assemblers are hardware-specific and are included implicitly as part of the compilers in the Single UNIX Specification.
- banner* The only known use of this command is as part of the *lp* printer header pages. It was decided that the format of the header is implementation-defined, so this utility is superfluous to applications portability.
- calendar* This reminder service program is not useful to conforming applications.
- cancel* The *lp* (line printer spooling) system specified is the most basic possible and did not need this level of application control.

<i>chroot</i>	This is primarily of administrative use, requiring superuser privileges.
<i>col</i>	No utilities defined in the Single UNIX Specification produce output requiring such a filter. The <i>nroff</i> text formatter is present on many historical systems and will continue to remain as an extension; <i>col</i> is expected to be shipped by all the systems that ship <i>nroff</i> .
<i>cpio</i>	This has been replaced by <i>pax</i> , for reasons explained in the rationale for that utility.
<i>cpp</i>	This is subsumed by <i>c99</i> .
<i>cu</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs.
<i>dc</i>	The functionality of this utility can be provided by the <i>bc</i> utility; <i>bc</i> was selected because it was easier to use and had superior functionality. Although the historical versions of <i>bc</i> are implemented using <i>dc</i> as a base, the Single UNIX Specification prescribes the interface and not the underlying mechanism used to implement it.
<i>dircmp</i>	Although a useful concept, the historical output of this directory comparison program is not suitable for processing in application programs. Also, the <i>diff -r</i> command gives equivalent functionality.
<i>dis</i>	Disassemblers are hardware-specific.
<i>emacs</i>	The community of <i>emacs</i> editing enthusiasts was adamant that the full <i>emacs</i> editor not be included in the standard because they were concerned that an attempt to standardize this very powerful environment would encourage vendors to ship versions conforming strictly to the standard, but lacking the extensibility required by the community. The author of the original <i>emacs</i> program also expressed his desire to omit the program. Furthermore, there were a number of historical UNIX systems that did not include <i>emacs</i> , or included it without supporting it, but there were very few that did not include and support <i>vi</i> .
<i>ld</i>	This is subsumed by <i>c99</i> .
<i>line</i>	The functionality of <i>line</i> can be provided with <i>read</i> .
<i>lint</i>	<p>This technology is partially subsumed by <i>c99</i>. It is also hard to specify the degree of checking for possible error conditions in programs in any compiler, and specifying what <i>lint</i> would do in these cases is equally difficult.</p> <p>It is fairly easy to specify what a compiler does. It requires specifying the language, what it does with that language, and stating that the interpretation of any incorrect program is unspecified. Unfortunately, any description of <i>lint</i> is required to specify what to do with erroneous programs. Since the number of possible errors and questionable programming practices is infinite, <i>lint</i> cannot be required to detect all errors of any given class.</p> <p>Additionally, some vendors complained that since many compilers are distributed in a binary form without a <i>lint</i> facility (because the ISO C standard does not require one), implementing the standard as a stand-alone product will be much harder. Rather than being able to build upon a standard compiler component (simply by providing <i>c99</i> as an interface), source to that compiler would most likely need to be modified to provide the <i>lint</i> functionality. This was considered a major burden on system providers for a very small gain to developers (users).</p>
<i>login</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs.

<i>lorder</i>	This utility is an aid in creating an implementation-defined detail of object libraries that the standard developers did not feel required standardization.
<i>lpstat</i>	The <i>lp</i> system specified is the most basic possible and did not need this level of application control.
<i>mail</i>	This utility was omitted in favor of <i>mailx</i> because there was a considerable functionality overlap between the two.
<i>mknod</i>	This was omitted in favor of <i>mkfifo</i> , as <i>mknod</i> has too many implementation-defined functions.
<i>news</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs.
<i>pack</i>	This compression program was considered inferior to <i>compress</i> .
<i>passwd</i>	This utility was proposed in a historical draft but met with too many objections to be included. There were various reasons: <ul style="list-style-type: none"> ■ Changing a password should not be viewed as a command, but as part of the login sequence. Changing a password should only be done while a trusted path is in effect. ■ Even though the text in early drafts was intended to allow a variety of implementations to conform, the security policy for one site may differ from another site running with identical hardware and software. One site might use password authentication while the other did not. Vendors could not supply a <i>passwd</i> utility that would conform to the Single UNIX Specification for all sites using their system. ■ This is really a subject for a system administration working group or a security working group.
<i>pcat</i>	This compression program was considered inferior to <i>zcat</i> .
<i>pg</i>	This duplicated many of the features of the <i>more</i> pager, which was preferred by the standard developers.
<i>prof</i>	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool.
RCS	RCS was originally considered as part of a version control utilities portion of the scope. However, this aspect was abandoned by the standard developers. SCCS is included as part of the Development option within the Single UNIX Specification.
<i>red</i>	Restricted editor. This was not considered by the standard developers because it never provided the level of security restriction required.
<i>rsh</i>	Restricted shell. This was not considered by the standard developers because it does not provide the level of security restriction that is implied by historical documentation.
<i>sdb</i>	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool. Furthermore, some useful aspects of <i>sdb</i> are very hardware-specific.

<i>sdiff</i>	The “side-by-side <i>diff</i> ” utility from System V was omitted because it is used infrequently, and even less so by conforming applications. Despite being in System V, it is not in the SVID or XPG.
<i>shar</i>	Any of the numerous “shell archivers” were excluded because they did not meet the requirement of existing practice.
<i>shl</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs. The job control aspects of the shell command language are generally more useful.
<i>size</i>	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool.
<i>spell</i>	This utility is not useful from shell scripts or typical application programs. The <i>spell</i> utility was considered, but was omitted because there is no known technology that can be used to make it recognize general language for user-specified input without providing a complete dictionary along with the input file.
<i>su</i>	This utility is not useful from shell scripts or typical application programs. (There was also sentiment to avoid security-related utilities.)
<i>sum</i>	This utility was renamed <i>cksum</i> .
<i>tar</i>	This utility has been replaced by <i>pax</i> , for reasons explained in the rationale for that utility.
<i>unpack</i>	This compression program was considered inferior to <i>uncompress</i> .
<i>wall</i>	This utility is terminal-oriented and is not useful in shell scripts or typical applications. It is generally used only by system administrators.

Headers

This chapter covers the header and name space rules, the scope of the headers, and an overview of the changes to them in this version of the specification.

5.1 Header and Name Space Rules

The headers in the Single UNIX Specification are built upon the headers from the ISO C standard.

5.1.1 ISO C Headers

The ISO C standard describes general rules for implementations *versus* applications with respect to headers and name space.

Headers

1. Headers need not be regular text files and the characters between the '`<`' and '`>`' need not name a source file.
2. Headers should be self-sufficient such that any standard header does not need any other standard header to be included using **#include** before or after it (and thus headers can be included in any order).
3. Headers should be idempotent, such that any standard header can be included any number of times without causing problems.
4. Headers can provide macro versions of functions which should behave in a semantically identical fashion. By undefining such a macro, a regular declaration of the function is made visible.
5. Headers can assume that they are included outside of any file scope declaration or definition.
6. Headers can assume that they are included prior to the first use of its functions, objects, or macros.

With the exception of **<assert.h>**, the ISO C standard headers are both self-sufficient and idempotent.

The ISO C standard specifies the rules for its 24 standard headers. Items 2. through 6. listed above for headers are requirements for these.

<assert.h>	<float.h>	<math.h>	<stddef.h>	<time.h>
<complex.h>	<inttypes.h>	<setjmp.h>	<stdio.h>	<tgmath.h>
<ctype.h>	<iso646.h>	<signal.h>	<stdint.h>	<wchar.h>
<errno.h>	<limits.h>	<stdarg.h>	<stdlib.h>	<wctype.h>
<fenv.h>	<locale.h>	<stdbool.h>	<string.h>	

A compilation system can provide more headers, but a strictly conforming ISO C standard program can only use these.

Name Space

1. `_[A-Z][0-9a-z_A-Z]*` are reserved for implementation use anywhere.
2. `_[0-9a-z_A-Z]*` are reserved for implementation use as identifiers with external linkage and in any header as a tag or an ordinary identifier with file scope.

5.1.2 POSIX.1-2008 Base Headers

The requirements for the POSIX.1-2008 base standard are a subset of the requirements for the Single UNIX Specification. For the purposes of this description, the POSIX.1-2008 base standard is the set of requirements excluding the XSI option.

POSIX.1-2008 specifies general rules for name space control in headers through *feature test macros*:

1. Feature test macros have names that match `_[A-Z][0-9a-z_A-Z]*`. (Thus, the use of such names takes advantage of the name space rules in the ISO C standard.) They almost always match `_[A-Z]+_SOURCE`. The original feature test macro for the POSIX.1-1990 standard was `_POSIX_SOURCE`. Later revisions of POSIX.1 use the more powerful feature test macro called `_POSIX_C_SOURCE`, which allows you to specify the revision of POSIX to which your application conforms. The value of `_POSIX_C_SOURCE` which corresponds to the functionality contained in the Single UNIX Specification, Version 4 is 200809L.
2. If the feature test macro associated with a particular header is defined prior to its inclusion, the header must obey the name space rules of the matching specification. For example, if the macro `_POSIX_C_SOURCE` is defined with the value 200809L before a POSIX standard header is included, the header makes visible only the symbols required by the standard and additional symbols explicitly permitted by the standard (including those in reserved name spaces). No other symbols are permitted to be visible (unless enabled by another feature test macro).
3. If such a header is included without any feature test macros being defined, it has no restrictions (at least with respect to the associated POSIX standard; headers defined by the ISO C standard make visible only the symbols allowed by the ISO C standard).

POSIX.1 specifies general rules for its headers:

1. `[a-z_A-Z][0-9a-z_A-Z]*_t` are reserved for all POSIX.1 headers.
2. `POSIX_*`, `_POSIX_*` and `posix_*` are reserved for use by POSIX.1 and other POSIX standards.
3. Each function must be declared with a prototype in at least one header.
4. The default location for this declaration is `<unistd.h>`.

POSIX.1 specifies the rules for the standard POSIX base headers and subsumes the headers

from the ISO C standard, giving additional rules to six of them:

<aio.h>	<monetary.h>	<sched.h>	<sys/un.h>
<arpa/inet.h>	<mqueue.h>	<semaphore.h>	<sys/utsname.h>
<cpio.h>	<net/if.h>	<spawn.h>	<sys/wait.h>
<dirent.h>	<netdb.h>	<strings.h>	<tar.h>
<dlfcn.h>	<netinet/in.h>	<sys/mman.h>	<termios.h>
<fcntl.h>	<netinet/tcp.h>	<sys/select.h>	<trace.h>
<fnmatch.h>	<nl_types.h>	<sys/socket.h>	<unistd.h>
<glob.h>	<poll.h>	<sys/stat.h>	<utime.h>
<grp.h>	<pthread.h>	<sys/statvfs.h>	<wordexp.h>
<iconv.h>	<pwd.h>	<sys/times.h>	
<langinfo.h>	<regex.h>	<sys/types.h>	

The six modified ISO C standard headers are:

```
<errno.h>
<limits.h>
<setjmp.h>
<signal.h>
<stdio.h>
<time.h>
```

5.1.3 XSI Headers

The XSI option adds the following headers:

<fmtmsg.h>	<search.h>	<sys/sem.h>	<sys/uiio.h>
<ftw.h>	<sys/ipc.h>	<sys/shm.h>	<syslog.h>
<libgen.h>	<sys/msg.h>	<sys/time.h>	<ulimit.h>
<ndbm.h>	<sys/resource.h>	<sys/timeb.h>	<utmpx.h>

The XSI STREAMS option adds the **<stropts.h>** header.

A feature test macro named `_XOPEN_SOURCE` provides for all the XSI extensions (which include the POSIX.1 base) to the ISO C standard, in a similar way to the `_POSIX_C_SOURCE` macro. For conformance to the Single UNIX Specification, Version 4, `_XOPEN_SOURCE` should be set to the value 700. The definition of `_XOPEN_SOURCE` subsumes the use of `_POSIX_C_SOURCE`, ensuring that the appropriate POSIX name space is also exposed.

5.2 Names Safe to Use

Reserved name rules are complicated and are set out in full in XSH, Section 2.2. There are, however, three fairly simple rules which, if followed, make collisions with any reserved names very unlikely:

1. Use **#include** to include all system headers at the top of source files.
2. Do not define or declare any names that begin with an underscore.
3. Use mixed case for all application-defined identifiers. POSIX.1 uses only uppercase letters in constants and lowercase letters in other identifiers, and implementations usually follow the same pattern for additional names they make visible in reserved name spaces.

Most implementations continue to add names to the standard headers. The `_POSIX_C_SOURCE` and `_XOPEN_SOURCE` macros can be used to remove the added symbols with unreserved names, so that a predictable standard name space is visible.

5.3 Base Documents

The headers provided in XBD, Issue 7 are drawn from the following base documents:

- IEEE Std 1003.1, 2004 Edition (POSIX-1) (incorporating IEEE Std 1003.1-2001, IEEE Std 1003.1-2001/Cor 1-2002, and IEEE Std 1003.1-2001/Cor 2-2004)
- The Open Group Technical Standard, 2006, Extended API Set Part 1
- The Open Group Technical Standard, 2006, Extended API Set Part 2
- The Open Group Technical Standard, 2006, Extended API Set Part 3
- The Open Group Technical Standard, 2006, Extended API Set Part 4
- ISO/IEC 9899:1999, Programming Languages — C (including ISO/IEC 9899:1999/Cor.1:2001(E), ISO/IEC 9899:1999/Cor.2:2004(E), and ISO/IEC 9899:1999/Cor.3(E))

5.4 Overview of Changes

The following sections describe changes made to the headers since Issue 6. [Chapter 13](#) (on page 341) details the technical changes that have been made to the headers from Issue 6.

Changes from Issue 6 to Issue 7

The following list summarizes the major changes that were made in the headers from Issue 6 to Issue 7:

- The Open Group Technical Standard, 2006, Extended API Set Part 1 is incorporated.
- The Open Group Technical Standard, 2006, Extended API Set Part 2 is incorporated.
- The Open Group Technical Standard, 2006, Extended API Set Part 3 is incorporated.
- The Open Group Technical Standard, 2006, Extended API Set Part 4 is incorporated.

- Existing functionality is aligned with ISO/IEC 9899: 1999, Programming Languages — C, ISO/IEC 9899: 1999/Cor.2: 2004(E).
- Austin Group defect reports, IEEE Interpretations against IEEE Std 1003.1, and responses to ISO/IEC defect reports against ISO/IEC 9945 are applied.
- The Open Group corrigenda and resolutions are applied.
- Issues arising from ISO TR 24715: 2006, Conflicts between POSIX and the LSB have been addressed.
- Features, marked legacy or obsolescent in the base document, have been considered for removal in this version. This includes removal of the **<ucontext.h>** header.
- The following headers which were optional in Issue 6 are now mandatory in Issue 7:
<aio.h>, **<pthread.h>**, **<sched.h>**, **<semaphore.h>**, **<sys/mman.h>**
- The following headers moved from the XSI option to the POSIX Base in Issue 7:
<cpio.h>, **<dlfcn.h>**, **<iconv.h>**, **<langinfo.h>**, **<monetary.h>**, **<nl_types.h>**, **<poll.h>**, **<strings.h>**, **<sys/statvfs.h>**
- The following headers have been marked obsolescent in Issue 7:
<stropts.h>, **<trace.h>**, **<ulimit.h>**, **<utime.h>**
- The header reference pages have been clarified with respect to macros and constants. In particular, a definition of the term *symbolic constant* has been added, with associated general rules which apply to every symbolic constant unless stated otherwise:
 - It expands to a compile-time constant expression with an integer type.
 - It may be defined as another type of constant—e.g., an enumeration constant—as well as being a macro.
 - It need not be usable in **#if** preprocessing directives.

In addition, wording changes have been made to ensure “macro” is used where constants are required to be defined only as macros, and explicit requirements have been added where macros are required to be usable in **#if** preprocessing directives.

New Features in Issue 7

There are no new headers in Issue 7.

Terminal Interfaces

The Single UNIX Specification, Version 4 contains X/Open Curses, Issue 7. These interfaces support a terminal-independent character screen update method. X/Open Curses, Issue 7 is the first update of X/Open Curses since Issue 4, Version 2. The Single UNIX Specification, Versions 1 through 3 all contained X/Open Curses, Issue 4, Version 2, with corrigenda specifying how it functioned in a Base Specifications Issue 5 and Issue 6 environment.

This chapter covers the contents of X/Open Curses, including a functional overview and an overview of the changes in this version.

6.1 Functional Overview

6.1.1 Curses Interfaces

X/Open Curses, Issue 7, Chapters 1 to 3, provide all the introductory material needed to work with the X/Open Curses interfaces. These interfaces include:

<i>COLORS()</i>	<i>delscreen()</i>	<i>insch()</i>	<i>restartterm()</i>	<i>unctrl()</i>
<i>COLOR_PAIR()</i>	<i>delwin()</i>	<i>insdelln()</i>	<i>ripoffline()</i>	<i>unget_wch()</i>
<i>COLOR_PAIRS()</i>	<i>derwin()</i>	<i>insertln()</i>	<i>savetty()</i>	<i>ungetch()</i>
<i>COLS()</i>	<i>doupdate()</i>	<i>insnstr()</i>	<i>scanw()</i>	<i>untouchwin()</i>
<i>LINES()</i>	<i>dupwin()</i>	<i>insstr()</i>	<i>scr_dump()</i>	<i>use_env()</i>
<i>PAIR_NUMBER()</i>	<i>echo()</i>	<i>instr()</i>	<i>scr_init()</i>	<i>vid_attr()</i>
<i>add_wch()</i>	<i>echo_wchar()</i>	<i>intrflush()</i>	<i>scr_restore()</i>	<i>vid_puts()</i>
<i>add_wchnstr()</i>	<i>echochar()</i>	<i>inwstr()</i>	<i>scr_set()</i>	<i>vidattr()</i>
<i>add_wchstr()</i>	<i>endwin()</i>	<i>is_linetouched()</i>	<i>scrll()</i>	<i>vidputs()</i>
<i>addch()</i>	<i>erase()</i>	<i>is_wintouched()</i>	<i>scroll()</i>	<i>vline()</i>
<i>addchnstr()</i>	<i>erasechar()</i>	<i>isendwin()</i>	<i>scrollok()</i>	<i>vline_set()</i>
<i>addchstr()</i>	<i>erasewchar()</i>	<i>key_name()</i>	<i>set_curterm()</i>	<i>vw_printw()</i>
<i>addnstr()</i>	<i>filter()</i>	<i>keyname()</i>	<i>set_term()</i>	<i>vw_scanw()</i>
<i>addnwstr()</i>	<i>flash()</i>	<i>keypad()</i>	<i>setcchar()</i>	<i>wattr_get()</i>
<i>addstr()</i>	<i>flushinp()</i>	<i>killchar()</i>	<i>setscrreg()</i>	<i>wattr_off()</i>
<i>addwstr()</i>	<i>get_wch()</i>	<i>killwchar()</i>	<i>setupterm()</i>	<i>wattr_on()</i>
<i>attr_get()</i>	<i>get_wstr()</i>	<i>leaveok()</i>	<i>slk_attr_off()</i>	<i>wattr_set()</i>
<i>attr_off()</i>	<i>getbegyx()</i>	<i>longname()</i>	<i>slk_attr_on()</i>	<i>wattroff()</i>
<i>attr_on()</i>	<i>getbkgd()</i>	<i>meta()</i>	<i>slk_attr_set()</i>	<i>wattron()</i>
<i>attr_set()</i>	<i>getbkgrnd()</i>	<i>move()</i>	<i>slk_attroff()</i>	<i>wattrset()</i>
<i>attroff()</i>	<i>getcchar()</i>	<i>mvcur()</i>	<i>slk_attron()</i>	<i>wbkgd()</i>
<i>attron()</i>	<i>getch()</i>	<i>mvderwin()</i>	<i>slk_attrset()</i>	<i>wbkgdset()</i>
<i>attrset()</i>	<i>getmaxyx()</i>	<i>mvwin()</i>	<i>slk_clear()</i>	<i>wbkgnd()</i>
<i>baudrate()</i>	<i>getn_wstr()</i>	<i>napms()</i>	<i>slk_color()</i>	<i>wbkgndset()</i>
<i>beep()</i>	<i>getnstr()</i>	<i>newpad()</i>	<i>slk_init()</i>	<i>wborder()</i>
<i>bkgd()</i>	<i>getparyx()</i>	<i>newterm()</i>	<i>slk_label()</i>	<i>wborder_set()</i>
<i>bkgdset()</i>	<i>getstr()</i>	<i>newwin()</i>	<i>slk_noutrefresh()</i>	<i>wclear()</i>
<i>bkgrnd()</i>	<i>getwin()</i>	<i>nl()</i>	<i>slk_refresh()</i>	<i>wclrtobot()</i>
<i>bkgrndset()</i>	<i>getyx()</i>	<i>nocbreak()</i>	<i>slk_restore()</i>	<i>wclrtoeol()</i>
<i>border()</i>	<i>halfdelay()</i>	<i>nodelay()</i>	<i>slk_set()</i>	<i>wcolor_set()</i>
<i>border_set()</i>	<i>has_colors()</i>	<i>noecho()</i>	<i>slk_touch()</i>	<i>wcolorsyncup()</i>
<i>box()</i>	<i>has_ic()</i>	<i>nonl()</i>	<i>slk_wset()</i>	<i>wdeleteln()</i>
<i>box_set()</i>	<i>has_il()</i>	<i>noqiflush()</i>	<i>standend()</i>	<i>wecho_wchar()</i>
<i>can_change_color()</i>	<i>hline()</i>	<i>noraw()</i>	<i>standout()</i>	<i>wechochar()</i>
<i>cbreak()</i>	<i>hline_set()</i>	<i>notimeout()</i>	<i>start_color()</i>	<i>werase()</i>
<i>chgat()</i>	<i>idcok()</i>	<i>overlay()</i>	<i>stdscr()</i>	<i>wgetbkgrnd()</i>
<i>clear()</i>	<i>idllok()</i>	<i>overwrite()</i>	<i>subpad()</i>	<i>winsdelln()</i>
<i>clearok()</i>	<i>immedok()</i>	<i>pair_content()</i>	<i>subwin()</i>	<i>winsertln()</i>
<i>clrtoBOT()</i>	<i>in_wch()</i>	<i>pecho_wchar()</i>	<i>syncok()</i>	<i>wmove()</i>
<i>clrtoeol()</i>	<i>in_wchnstr()</i>	<i>pechochar()</i>	<i>term_attrs()</i>	<i>wnoutrefresh()</i>
<i>color_content()</i>	<i>in_wchstr()</i>	<i>pnoutrefresh()</i>	<i>termattrs()</i>	<i>wredrawln()</i>
<i>color_set()</i>	<i>inch()</i>	<i>prefresh()</i>	<i>termname()</i>	<i>wrefresh()</i>
<i>copywin()</i>	<i>inchnstr()</i>	<i>printw()</i>	<i>tigetflag()</i>	<i>wscrll()</i>
<i>cur_term()</i>	<i>inchstr()</i>	<i>putp()</i>	<i>tigetnum()</i>	<i>wsetscrreg()</i>
<i>curs_set()</i>	<i>init_color()</i>	<i>putwin()</i>	<i>tigetstr()</i>	<i>wstandend()</i>
<i>curscr()</i>	<i>init_pair()</i>	<i>qiflush()</i>	<i>timeout()</i>	<i>wstandout()</i>
<i>def_prog_mode()</i>	<i>initscr()</i>	<i>raw()</i>	<i>tiparm()</i>	<i>wsyncdown()</i>
<i>def_shell_mode()</i>	<i>innstr()</i>	<i>redrawwin()</i>	<i>touchline()</i>	<i>wsyncup()</i>
<i>del_curterm()</i>	<i>innwstr()</i>	<i>refresh()</i>	<i>touchwin()</i>	<i>wtimeout()</i>
<i>delay_output()</i>	<i>ins_nwstr()</i>	<i>reset_prog_mode()</i>	<i>tparm()</i>	<i>wtouchln()</i>
<i>delch()</i>	<i>ins_wch()</i>	<i>reset_shell_mode()</i>	<i>tputs()</i>	<i>wunctrl()</i>
<i>deleteln()</i>	<i>ins_wstr()</i>	<i>resetty()</i>	<i>typeahead()</i>	

Note that there are functions listed in the table that represent a set of related functions. For example, *addch()* represents the set of functions *addch()*, *mvaddch()*, *mvwaddch()*, and *waddch()*. A table in *X/Open Curses*, Issue 7, Section 3.6.2, Function Families Provided, summarizes these families of related calls.

6.1.2 Curses Utilities

X/Open Curses, Issue 7 is the first version of *X/Open Curses* to include utilities. The utilities all relate to handling of the **terminfo** database:

infocmp, *tic*, *tput*, *untic*

The *tput* utility can be used in shell scripts to perform tasks similar to those for which *tiparm()* (or the obsolescent *tparm()*) is used in C programs.

6.2 Overview of Changes

The following sections describe changes made to *X/Open Curses* since Issue 4, Version 2. The CHANGE HISTORY section for each entry details the technical changes that have been made to that entry from Issue 4, Version 2.

Changes from Issue 4, Version 2 to Issue 7

Issue 7 is updated as follows:

- Functionality marked “To Be Withdrawn” is removed.
- Clarification is added to explain that the **int** arguments passed to *getbegyx()*, *getmaxyx()*, *getparyx()*, and *getyx()* must be modifiable lvalues.
- The *tparm()* function is marked obsolescent.
- Features described below are introduced.

Features Introduced in Issue 7

The following features are introduced in Issue 7:

- Function prototypes are updated to use **const** where appropriate.
- The *tiparm()* function is added.
- The following new utilities are added:

infocmp, *tic*, *tput*, *untic*

Features Withdrawn in Issue 7

The following interfaces are withdrawn in Issue 7:

Withdrawn Interfaces			
<i>tgetent()</i>	<i>tgetnum()</i>	<i>tgoto()</i>	<i>vwprintw()</i>
<i>tgetflag()</i>	<i>tgetstr()</i>	<i>vwscanw()</i>	

System Interface Table

7.1 Introduction

This chapter contains a table of all the interfaces defined in XSH, Issue 7, complete with an indication of their status for XSI-conforming and POSIX-conforming systems, and their availability in UNIX 03 (denoted by U03), UNIX 98 (denoted by U98), UNIX 95 (denoted by U95), POSIX.1-2001 (denoted P01), the ISO POSIX-1:1996 standard (denoted P96), the POSIX.2-1992 standard (denoted P92), the ISO C standard (denoted C99), and C89.

The following conventions are used in columns 2 through 10:

- m** Indicates that the interface is defined as mandatory.
- o** Indicates that the interface is part of an Option or Feature Group.
- ob** Indicates that the interface is Obsolescent, and although mandatory for the implementation, applications are discouraged from its use.
- ob o** In the XSI column, indicates that the interface is Obsolescent, and is part of an Option or Feature Group. Applications are discouraged from its use.
- l** In the U03 column, indicates that the interface is part of the Legacy Option Group and need not be available on all implementations.
- opt* In the POSIX and P01 columns, two or three letter option codes are used as described in XBD, Issue 7 (for the POSIX column) and XBD, Issue 6 (for the P01 column), denoting the option to which the interface belongs.
- ob opt** Same as *opt*, but the interface is also obsolescent.
- r** In the P96 column, indicates that the interface is part of the POSIX Realtime Extension.
- t** In the P96 column, indicates that the interface is part of the POSIX Threads Extension.
- 1** In the C89 column, indicates that the interface is part of the ISO/IEC 9899: 1990/Amendment 1: 1995 (E).
- .** Indicates that the interface is not specified.

The table is intended as a quick reference guide for programmers migrating to or developing applications for the Single UNIX Specification, Version 4. Products that brand to a profile may not provide all of the interfaces listed, depending on which Option or Feature Groups are supported.

7.2 System Interface Table

There are 1191 interfaces listed.

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>FD_CLR()</i>	m	m	m	m	m	m	.	.	.
<i>FD_ISSET()</i>	m	m	m	m	m	m	.	.	.
<i>FD_SET()</i>	m	m	m	m	m	m	.	.	.
<i>FD_ZERO()</i>	m	m	m	m	m	m	.	.	.
<i>_Exit()</i>	m	m	m	m	.	.	.	m	.
<i>_exit()</i>	m	m	m	m	m	m	m	.	.
<i>_longjmp()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>_setjmp()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>_tolower()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>_toupper()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>a64l()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>abort()</i>	m	m	m	m	m	m	m	m	m
<i>abs()</i>	m	m	m	m	m	m	m	m	m
<i>accept()</i>	m	m	m	m	m	m	.	.	.
<i>access()</i>	m	m	m	m	m	m	m	.	.
<i>acos()</i>	m	m	m	m	m	m	m	m	m
<i>acosf()</i>	m	m	m	m	.	.	.	m	.
<i>acosh()</i>	m	m	m	m	m	m	.	m	.
<i>acoshf()</i>	m	m	m	m	.	.	.	m	.
<i>acoshl()</i>	m	m	m	m	.	.	.	m	.
<i>acosl()</i>	m	m	m	m	.	.	.	m	.
<i>aio_cancel()</i>	m	m	o	aio	o	.	r	.	.
<i>aio_error()</i>	m	m	o	aio	o	.	r	.	.
<i>aio_fsync()</i>	m	m	o	aio	o	.	r	.	.
<i>aio_read()</i>	m	m	o	aio	o	.	r	.	.
<i>aio_return()</i>	m	m	o	aio	o	.	r	.	.
<i>aio_suspend()</i>	m	m	o	aio	o	.	r	.	.
<i>aio_write()</i>	m	m	o	aio	o	.	r	.	.
<i>alarm()</i>	m	m	m	m	m	m	m	.	.
<i>alphasort()</i>	m	m
<i>asctime()</i>	ob	ob	m	m	m	m	m	m	m
<i>asctime_r()</i>	ob	ob	m	tsf	m	.	t	.	.
<i>asin()</i>	m	m	m	m	m	m	m	m	m
<i>asinf()</i>	m	m	m	m	.	.	.	m	.
<i>asinh()</i>	m	m	m	m	m	m	.	m	.
<i>asinhf()</i>	m	m	m	m	.	.	.	m	.
<i>asinhf()</i>	m	m	m	m	.	.	.	m	.
<i>asinhf()</i>	m	m	m	m	.	.	.	m	.
<i>asinl()</i>	m	m	m	m	.	.	.	m	.
<i>assert()</i>	m	m	m	m	m	m	m	m	m
<i>atan2()</i>	m	m	m	m	m	m	m	m	m
<i>atan2f()</i>	m	m	m	m	.	.	.	m	.
<i>atan2l()</i>	m	m	m	m	.	.	.	m	.
<i>atan()</i>	m	m	m	m	m	m	m	m	m
<i>atanf()</i>	m	m	m	m	.	.	.	m	.
<i>atanh()</i>	m	m	m	m	m	m	.	m	.
<i>atanhf()</i>	m	m	m	m	.	.	.	m	.
<i>atanhf()</i>	m	m	m	m	.	.	.	m	.
<i>atanh()</i>	m	m	m	m	.	.	.	m	.
<i>atanl()</i>	m	m	m	m	.	.	.	m	.
<i>atexit()</i>	m	m	m	m	m	m	.	m	m
<i>atof()</i>	m	m	m	m	m	m	m	m	m

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>atoi()</i>	m	m	m	m	m	m	m	m	m
<i>atol()</i>	m	m	m	m	m	m	m	m	m
<i>atoll()</i>	m	m	m	m	.	.	.	m	.
<i>basename()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>bind()</i>	m	m	m	m	m	m	.	.	.
<i>bsearch()</i>	m	m	m	m	m	m	m	m	m
<i>btowc()</i>	m	m	m	m	m	.	.	m	1
<i>cabs()</i>	m	m	m	m	.	.	.	m	.
<i>cabsf()</i>	m	m	m	m	.	.	.	m	.
<i>cabsl()</i>	m	m	m	m	.	.	.	m	.
<i>cacos()</i>	m	m	m	m	.	.	.	m	.
<i>cacosf()</i>	m	m	m	m	.	.	.	m	.
<i>cacosh()</i>	m	m	m	m	.	.	.	m	.
<i>cacoshf()</i>	m	m	m	m	.	.	.	m	.
<i>cacoshl()</i>	m	m	m	m	.	.	.	m	.
<i>cacosl()</i>	m	m	m	m	.	.	.	m	.
<i>calloc()</i>	m	m	m	m	m	m	m	m	m
<i>carg()</i>	m	m	m	m	.	.	.	m	.
<i>cargf()</i>	m	m	m	m	.	.	.	m	.
<i>cargl()</i>	m	m	m	m	.	.	.	m	.
<i>casin()</i>	m	m	m	m	.	.	.	m	.
<i>casinf()</i>	m	m	m	m	.	.	.	m	.
<i>casinh()</i>	m	m	m	m	.	.	.	m	.
<i>casinhf()</i>	m	m	m	m	.	.	.	m	.
<i>casinhl()</i>	m	m	m	m	.	.	.	m	.
<i>casinl()</i>	m	m	m	m	.	.	.	m	.
<i>catan()</i>	m	m	m	m	.	.	.	m	.
<i>catanf()</i>	m	m	m	m	.	.	.	m	.
<i>catanh()</i>	m	m	m	m	.	.	.	m	.
<i>catanhf()</i>	m	m	m	m	.	.	.	m	.
<i>catanhl()</i>	m	m	m	m	.	.	.	m	.
<i>catanl()</i>	m	m	m	m	.	.	.	m	.
<i>catclose()</i>	m	m	m	xsi	m	m	.	.	.
<i>catgets()</i>	m	m	m	xsi	m	m	.	.	.
<i>catopen()</i>	m	m	m	xsi	m	m	.	.	.
<i>cbrt()</i>	m	m	m	m	m	m	.	m	.
<i>cbrtf()</i>	m	m	m	m	.	.	.	m	.
<i>cbrtl()</i>	m	m	m	m	.	.	.	m	.
<i>ccos()</i>	m	m	m	m	.	.	.	m	.
<i>ccosf()</i>	m	m	m	m	.	.	.	m	.
<i>ccosh()</i>	m	m	m	m	.	.	.	m	.
<i>ccoshf()</i>	m	m	m	m	.	.	.	m	.
<i>ccoshl()</i>	m	m	m	m	.	.	.	m	.
<i>ccosl()</i>	m	m	m	m	.	.	.	m	.
<i>ceil()</i>	m	m	m	m	m	m	m	m	m
<i>ceilf()</i>	m	m	m	m	.	.	.	m	.
<i>ceill()</i>	m	m	m	m	.	.	.	m	.
<i>cexp()</i>	m	m	m	m	.	.	.	m	.
<i>cexpf()</i>	m	m	m	m	.	.	.	m	.
<i>cexpl()</i>	m	m	m	m	.	.	.	m	.

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>cfgetispeed()</i>	m	m	m	m	m	m	m	.	.
<i>cfgetospeed()</i>	m	m	m	m	m	m	m	.	.
<i>cfsetispeed()</i>	m	m	m	m	m	m	m	.	.
<i>cfsetospeed()</i>	m	m	m	m	m	m	m	.	.
<i>chdir()</i>	m	m	m	m	m	m	m	.	.
<i>chmod()</i>	m	m	m	m	m	m	m	.	.
<i>chown()</i>	m	m	m	m	m	m	m	.	.
<i>cimag()</i>	m	m	m	m	.	.	.	m	.
<i>cimagf()</i>	m	m	m	m	.	.	.	m	.
<i>cimagl()</i>	m	m	m	m	.	.	.	m	.
<i>clearerr()</i>	m	m	m	m	m	m	m	m	m
<i>clock()</i>	m	m	m	m	m	m	.	m	m
<i>clock_getcpuclockid()</i>	o	cpt	o	cpt
<i>clock_getres()</i>	m	m	o	tmr	o	.	r	.	.
<i>clock_gettime()</i>	m	m	o	tmr	o	.	r	.	.
<i>clock_nanosleep()</i>	m	m	o	cs
<i>clock_settime()</i>	m	m	o	tmr	o	.	r	.	.
<i>clog()</i>	m	m	m	m	.	.	.	m	.
<i>clogf()</i>	m	m	m	m	.	.	.	m	.
<i>clogl()</i>	m	m	m	m	.	.	.	m	.
<i>close()</i>	m	m	m	m	m	m	m	.	.
<i>closedir()</i>	m	m	m	m	m	m	m	.	.
<i>closelog()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>confstr()</i>	m	m	m	m	m	m	m	.	.
<i>conj()</i>	m	m	m	m	.	.	.	m	.
<i>conjf()</i>	m	m	m	m	.	.	.	m	.
<i>conjl()</i>	m	m	m	m	.	.	.	m	.
<i>connect()</i>	m	m	m	m	m	m	.	.	.
<i>copysign()</i>	m	m	m	m	.	.	.	m	.
<i>copysignf()</i>	m	m	m	m	.	.	.	m	.
<i>copysignl()</i>	m	m	m	m	.	.	.	m	.
<i>cos()</i>	m	m	m	m	m	m	m	m	m
<i>cosf()</i>	m	m	m	m	.	.	.	m	.
<i>cosh()</i>	m	m	m	m	m	m	m	m	m
<i>coshf()</i>	m	m	m	m	.	.	.	m	.
<i>coshl()</i>	m	m	m	m	.	.	.	m	.
<i>cosl()</i>	m	m	m	m	.	.	.	m	.
<i>cpow()</i>	m	m	m	m	.	.	.	m	.
<i>cpowf()</i>	m	m	m	m	.	.	.	m	.
<i>cpowl()</i>	m	m	m	m	.	.	.	m	.
<i>cproj()</i>	m	m	m	m	.	.	.	m	.
<i>cprojf()</i>	m	m	m	m	.	.	.	m	.
<i>cprojl()</i>	m	m	m	m	.	.	.	m	.
<i>creal()</i>	m	m	m	m	.	.	.	m	.
<i>crealf()</i>	m	m	m	m	.	.	.	m	.
<i>creall()</i>	m	m	m	m	.	.	.	m	.
<i>creat()</i>	m	m	m	m	m	m	m	.	.
<i>crypt()</i>	m	xsi	o	xsi	o	o	.	.	.
<i>csin()</i>	m	m	m	m	.	.	.	m	.
<i>csinf()</i>	m	m	m	m	.	.	.	m	.

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>csinh()</i>	m	m	m	m	.	.	.	m	.
<i>csinhf()</i>	m	m	m	m	.	.	.	m	.
<i>csinhl()</i>	m	m	m	m	.	.	.	m	.
<i>csinl()</i>	m	m	m	m	.	.	.	m	.
<i>csqrt()</i>	m	m	m	m	.	.	.	m	.
<i>csqrtf()</i>	m	m	m	m	.	.	.	m	.
<i>csqrtl()</i>	m	m	m	m	.	.	.	m	.
<i>ctan()</i>	m	m	m	m	.	.	.	m	.
<i>ctanf()</i>	m	m	m	m	.	.	.	m	.
<i>ctanh()</i>	m	m	m	m	.	.	.	m	.
<i>ctanhf()</i>	m	m	m	m	.	.	.	m	.
<i>ctanhl()</i>	m	m	m	m	.	.	.	m	.
<i>ctanl()</i>	m	m	m	m	.	.	.	m	.
<i>ctermid()</i>	m	m	m	m	m	m	m	.	.
<i>ctime()</i>	ob	ob	m	m	m	m	m	m	m
<i>ctime_r()</i>	ob	ob	m	tsf	m	.	t	.	.
<i>daylight</i>	m	xsi	m	xsi	m	m	.	.	.
<i>dbm_clearerr()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>dbm_close()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>dbm_delete()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>dbm_error()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>dbm_fetch()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>dbm_firstkey()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>dbm_nextkey()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>dbm_open()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>dbm_store()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>difftime()</i>	m	m	m	m	m	m	.	m	m
<i>dirfd()</i>	m	m
<i>dirname()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>div()</i>	m	m	m	m	m	m	.	m	m
<i>dlclose()</i>	m	m	m	xsi	m
<i>dlderror()</i>	m	m	m	xsi	m
<i>dlopen()</i>	m	m	m	xsi	m
<i>dlsym()</i>	m	m	m	xsi	m
<i>dprintf()</i>	m	m
<i>drand48()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>dup2()</i>	m	m	m	m	m	m	m	.	.
<i>dup()</i>	m	m	m	m	m	m	m	.	.
<i>duplocale()</i>	m	m
<i>encrypt()</i>	m	xsi	o	xsi	o	o	.	.	.
<i>endgrent()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>endhostent()</i>	m	m	m	m	m	m	.	.	.
<i>endnetent()</i>	m	m	m	m	m	m	.	.	.
<i>endprotoent()</i>	m	m	m	m	m	m	.	.	.
<i>endpwent()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>endservent()</i>	m	m	m	m	m	m	.	.	.
<i>endutxent()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>environ</i>	m	m	m	m	m	m	m	.	.
<i>erand48()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>erf()</i>	m	m	m	m	m	m	.	m	.

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>erfc()</i>	m	m	m	m	m	m	.	m	.
<i>erfcf()</i>	m	m	m	m	.	.	.	m	.
<i>erfcl()</i>	m	m	m	m	.	.	.	m	.
<i>erff()</i>	m	m	m	m	.	.	.	m	.
<i>erfl()</i>	m	m	m	m	.	.	.	m	.
<i>errno</i>	m	m	m	m	m	m	m	m	m
<i>execl()</i>	m	m	m	m	m	m	m	.	.
<i>execle()</i>	m	m	m	m	m	m	m	.	.
<i>execlp()</i>	m	m	m	m	m	m	m	.	.
<i>execv()</i>	m	m	m	m	m	m	m	.	.
<i>execve()</i>	m	m	m	m	m	m	m	.	.
<i>execvp()</i>	m	m	m	m	m	m	m	.	.
<i>exit()</i>	m	m	m	m	m	m	m	m	m
<i>exp2()</i>	m	m	m	m	.	.	.	m	.
<i>exp2f()</i>	m	m	m	m	.	.	.	m	.
<i>exp2l()</i>	m	m	m	m	.	.	.	m	.
<i>exp()</i>	m	m	m	m	m	m	m	m	m
<i>expf()</i>	m	m	m	m	.	.	.	m	.
<i>expl()</i>	m	m	m	m	.	.	.	m	.
<i>expm1()</i>	m	m	m	m	m	m	.	m	.
<i>expm1f()</i>	m	m	m	m	.	.	.	m	.
<i>expm1l()</i>	m	m	m	m	.	.	.	m	.
<i>fabs()</i>	m	m	m	m	m	m	m	m	m
<i>fabsf()</i>	m	m	m	m	.	.	.	m	.
<i>fabsl()</i>	m	m	m	m	.	.	.	m	.
<i>faccessat()</i>	m	m
<i>fattach()</i>	ob o	ob xsr	o	xsr	m	m	.	.	.
<i>fchdir()</i>	m	m	m	xsi	m	m	.	.	.
<i>fchmod()</i>	m	m	m	m	m	m	.	.	.
<i>fchmodat()</i>	m	m
<i>fchown()</i>	m	m	m	m	m	m	.	.	.
<i>fchownat()</i>	m	m
<i>fclose()</i>	m	m	m	m	m	m	m	m	m
<i>fcntl()</i>	m	m	m	m	m	m	m	.	.
<i>fdatasync()</i>	o	sio	o	sio	o	.	r	.	.
<i>fdetach()</i>	ob o	ob xsr	o	xsr	m	m	.	.	.
<i>fdim()</i>	m	m	m	m	.	.	.	m	.
<i>fdimf()</i>	m	m	m	m	.	.	.	m	.
<i>fdiml()</i>	m	m	m	m	.	.	.	m	.
<i>fdopen()</i>	m	m	m	m	m	m	m	.	.
<i>fdopendir()</i>	m	m
<i>feclearexcept()</i>	m	m	m	m	.	.	.	m	.
<i>fegetenv()</i>	m	m	m	m	.	.	.	m	.
<i>fegetexceptflag()</i>	m	m	m	m	.	.	.	m	.
<i>fegetround()</i>	m	m	m	m	.	.	.	m	.
<i>feholdexcept()</i>	m	m	m	m	.	.	.	m	.
<i>feof()</i>	m	m	m	m	m	m	m	m	m
<i>feraiseexcept()</i>	m	m	m	m	.	.	.	m	.
<i>ferror()</i>	m	m	m	m	m	m	m	m	m
<i>fesetenv()</i>	m	m	m	m	.	.	.	m	.

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>fesetexceptflag()</i>	m	m	m	m	.	.	.	m	.
<i>fesetround()</i>	m	m	m	m	.	.	.	m	.
<i>fetestexcept()</i>	m	m	m	m	.	.	.	m	.
<i>feupdateenv()</i>	m	m	m	m	.	.	.	m	.
<i>fexecve()</i>	m	m
<i>fflush()</i>	m	m	m	m	m	m	m	m	m
<i>ffs()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>fgetc()</i>	m	m	m	m	m	m	m	m	m
<i>fgetpos()</i>	m	m	m	m	m	m	.	m	m
<i>fgets()</i>	m	m	m	m	m	m	m	m	m
<i>fgetwc()</i>	m	m	m	m	m	m	.	m	1
<i>fgetws()</i>	m	m	m	m	m	m	.	m	1
<i>fileno()</i>	m	m	m	m	m	m	m	.	.
<i>flockfile()</i>	m	m	m	tsf	m	.	t	.	.
<i>floor()</i>	m	m	m	m	m	m	m	m	m
<i>floorf()</i>	m	m	m	m	.	.	.	m	.
<i>floorl()</i>	m	m	m	m	.	.	.	m	.
<i>fma()</i>	m	m	m	m	.	.	.	m	.
<i>maf()</i>	m	m	m	m	.	.	.	m	.
<i>mal()</i>	m	m	m	m	.	.	.	m	.
<i>fmax()</i>	m	m	m	m	.	.	.	m	.
<i>fmaxf()</i>	m	m	m	m	.	.	.	m	.
<i>fmaxl()</i>	m	m	m	m	.	.	.	m	.
<i>fmemopen()</i>	m	m
<i>fmin()</i>	m	m	m	m	.	.	.	m	.
<i>fminf()</i>	m	m	m	m	.	.	.	m	.
<i>fminl()</i>	m	m	m	m	.	.	.	m	.
<i>fmod()</i>	m	m	m	m	m	m	m	m	m
<i>fmodf()</i>	m	m	m	m	.	.	.	m	.
<i>fmodl()</i>	m	m	m	m	.	.	.	m	.
<i>fmtmsg()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>fnmatch()</i>	m	m	m	m	m	m	m	.	.
<i>fopen()</i>	m	m	m	m	m	m	m	m	m
<i>fork()</i>	m	m	m	m	m	m	m	.	.
<i>fpathconf()</i>	m	m	m	m	m	m	m	.	.
<i>fpclassify()</i>	m	m	m	m	.	.	.	m	.
<i>fprintf()</i>	m	m	m	m	m	m	m	m	m
<i>fputc()</i>	m	m	m	m	m	m	m	m	m
<i>fputs()</i>	m	m	m	m	m	m	m	m	m
<i>fputwc()</i>	m	m	m	m	m	m	.	m	1
<i>fputws()</i>	m	m	m	m	m	m	.	m	1
<i>fread()</i>	m	m	m	m	m	m	m	m	m
<i>free()</i>	m	m	m	m	m	m	m	m	m
<i>freeaddrinfo()</i>	m	m	m	m
<i>freelocale()</i>	m	m
<i>freopen()</i>	m	m	m	m	m	m	m	m	m
<i>frexp()</i>	m	m	m	m	m	m	m	m	m
<i>frexpf()</i>	m	m	m	m	.	.	.	m	.
<i>frexpl()</i>	m	m	m	m	.	.	.	m	.
<i>fscanf()</i>	m	m	m	m	m	m	m	m	m

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>fseek()</i>	m	m	m	m	m	m	m	m	m
<i>fseeko()</i>	m	m	m	m	m
<i>fsetpos()</i>	m	m	m	m	m	m	.	m	m
<i>fstat()</i>	m	m	m	m	m	m	m	.	.
<i>fstatat()</i>	m	m
<i>fstatvfs()</i>	m	m	m	xsi	m	m	.	.	.
<i>fsync()</i>	m	fsc	m	fsc	m	m	.	.	.
<i>ftell()</i>	m	m	m	m	m	m	m	m	m
<i>ftello()</i>	m	m	m	m	m
<i>ftok()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>ftruncate()</i>	m	m	m	m	m	m	.	.	.
<i>ftrylockfile()</i>	m	m	m	tsf	m	.	t	.	.
<i>ftw()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>funlockfile()</i>	m	m	m	tsf	m	.	t	.	.
<i>futimens()</i>	m	m
<i>fwide()</i>	m	m	m	m	m	.	.	m	1
<i>fwprintf()</i>	m	m	m	m	m	.	.	m	1
<i>fwrite()</i>	m	m	m	m	m	m	m	m	m
<i>fwscanf()</i>	m	m	m	m	m	.	.	m	1
<i>gai_strerror()</i>	m	m	m	m
<i>getaddrinfo()</i>	m	m	m	m
<i>getc()</i>	m	m	m	m	m	m	m	m	m
<i>getc_unlocked()</i>	m	m	m	tsf	m	.	t	.	.
<i>getchar()</i>	m	m	m	m	m	m	m	m	m
<i>getchar_unlocked()</i>	m	m	m	tsf	m	.	t	.	.
<i>getcwd()</i>	m	m	m	m	m	m	m	.	.
<i>getdate()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>getdate_err</i>	m	xsi	m	xsi	m	m	.	.	.
<i>getdelim()</i>	m	m
<i>getegid()</i>	m	m	m	m	m	m	m	.	.
<i>getenv()</i>	m	m	m	m	m	m	m	m	m
<i>geteuid()</i>	m	m	m	m	m	m	m	.	.
<i>getgid()</i>	m	m	m	m	m	m	m	.	.
<i>getgrent()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>getgrgid()</i>	m	m	m	m	m	m	m	.	.
<i>getgrgid_r()</i>	m	m	m	tsf	m	.	t	.	.
<i>getgrnam()</i>	m	m	m	m	m	m	m	.	.
<i>getgrnam_r()</i>	m	m	m	tsf	m	.	t	.	.
<i>getgroups()</i>	m	m	m	m	m	m	m	.	.
<i>gethostent()</i>	m	m	m	m	m	m	.	.	.
<i>gethostid()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>gethostname()</i>	m	m	m	m	m	m	.	.	.
<i>getitimer()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>getline()</i>	m	m
<i>getlogin()</i>	m	m	m	m	m	m	m	.	.
<i>getlogin_r()</i>	m	m	m	tsf	m	.	t	.	.
<i>getmsg()</i>	ob o	ob xsr	o	xsr	m	m	.	.	.
<i>getnameinfo()</i>	m	m	m	m
<i>getnetbyaddr()</i>	m	m	m	m	m	m	.	.	.
<i>getnetbyname()</i>	m	m	m	m	m	m	.	.	.

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>getnetent()</i>	m	m	m	m	m	m	.	.	.
<i>getopt()</i>	m	m	m	m	m	m	m	.	.
<i>getpeername()</i>	m	m	m	m	m	m	.	.	.
<i>getpgid()</i>	m	m	m	xsi	m	m	.	.	.
<i>getpgrp()</i>	m	m	m	m	m	m	m	.	.
<i>getpid()</i>	m	m	m	m	m	m	m	.	.
<i>getpmsg()</i>	ob o	ob xsr	o	xsr	m	m	.	.	.
<i>getppid()</i>	m	m	m	m	m	m	m	.	.
<i>getpriority()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>getprotobyname()</i>	m	m	m	m	m	m	.	.	.
<i>getprotobynumber()</i>	m	m	m	m	m	m	.	.	.
<i>getprotoent()</i>	m	m	m	m	m	m	.	.	.
<i>getpwent()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>getpwnam()</i>	m	m	m	m	m	m	m	.	.
<i>getpwnam_r()</i>	m	m	m	tsf	m	.	t	.	.
<i>getpwuid()</i>	m	m	m	m	m	m	m	.	.
<i>getpwuid_r()</i>	m	m	m	tsf	m	.	t	.	.
<i>getrlimit()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>getrusage()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>gets()</i>	ob	ob	m	m	m	m	m	m	m
<i>getservbyname()</i>	m	m	m	m	m	m	.	.	.
<i>getservbyport()</i>	m	m	m	m	m	m	.	.	.
<i>getservent()</i>	m	m	m	m	m	m	.	.	.
<i>getsid()</i>	m	m	m	xsi	m	m	.	.	.
<i>getsockname()</i>	m	m	m	m	m	m	.	.	.
<i>getsockopt()</i>	m	m	m	m	m	m	.	.	.
<i>getsubopt()</i>	m	m	m	xsi	m	m	.	.	.
<i>gettimeofday()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>getuid()</i>	m	m	m	m	m	m	m	.	.
<i>getutxent()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>getutxid()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>getutxline()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>getwc()</i>	m	m	m	m	m	m	.	m	1
<i>getwchar()</i>	m	m	m	m	m	m	.	m	1
<i>glob()</i>	m	m	m	m	m	o	m	.	.
<i>globfree()</i>	m	m	m	m	m	o	m	.	.
<i>gmtime()</i>	m	m	m	m	m	m	m	m	m
<i>gmtime_r()</i>	m	m	m	tsf	m	.	t	.	.
<i>grantpt()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>hcreate()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>hdestroy()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>hsearch()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>htonl()</i>	m	m	m	m	m	m	.	.	.
<i>htons()</i>	m	m	m	m	m	m	.	.	.
<i>hypot()</i>	m	m	m	m	m	m	.	m	.
<i>hypotf()</i>	m	m	m	m	.	.	.	m	.
<i>hypotl()</i>	m	m	m	m	.	.	.	m	.
<i>iconv()</i>	m	m	m	xsi	m	m	.	.	.
<i>iconv_close()</i>	m	m	m	xsi	m	m	.	.	.
<i>iconv_open()</i>	m	m	m	xsi	m	m	.	.	.

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>if_freenameindex()</i>	m	m	m	m
<i>if_indextoname()</i>	m	m	m	m
<i>if_nameindex()</i>	m	m	m	m
<i>if_nametoindex()</i>	m	m	m	m
<i>ilogb()</i>	m	m	m	m	m	m	.	m	.
<i>ilogbf()</i>	m	m	m	m	.	.	.	m	.
<i>ilogbl()</i>	m	m	m	m	.	.	.	m	.
<i>imaxabs()</i>	m	m	m	m	.	.	.	m	.
<i>imaxdiv()</i>	m	m	m	m	.	.	.	m	.
<i>inet_addr()</i>	m	m	m	m	m	m	.	.	.
<i>inet_ntoa()</i>	m	m	m	m	m	m	.	.	.
<i>inet_ntop()</i>	m	m	m	m
<i>inet_pton()</i>	m	m	m	m
<i>initstate()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>insque()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>ioctl()</i>	ob o	ob xsr	o	xsr	m	m	.	.	.
<i>isalnum()</i>	m	m	m	m	m	m	m	m	m
<i>isalnum_I()</i>	m	m
<i>isalpha()</i>	m	m	m	m	m	m	m	m	m
<i>isalpha_I()</i>	m	m
<i>isascii()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>isastream()</i>	ob o	ob xsr	o	xsr	m	m	.	.	.
<i>isatty()</i>	m	m	m	m	m	m	m	.	.
<i>isblank()</i>	m	m	m	m	.	.	.	m	.
<i>isblank_I()</i>	m	m
<i>iscntrl()</i>	m	m	m	m	m	m	m	m	m
<i>iscntrl_I()</i>	m	m
<i>isdigit()</i>	m	m	m	m	m	m	m	m	m
<i>isdigit_I()</i>	m	m
<i>isfinite()</i>	m	m	m	m	.	.	.	m	.
<i>isgraph()</i>	m	m	m	m	m	m	m	m	m
<i>isgraph_I()</i>	m	m
<i>isgreater()</i>	m	m	m	m	.	.	.	m	.
<i>isgreaterequal()</i>	m	m	m	m	.	.	.	m	.
<i>isinf()</i>	m	m	m	m	.	.	.	m	.
<i>isless()</i>	m	m	m	m	.	.	.	m	.
<i>islessequal()</i>	m	m	m	m	.	.	.	m	.
<i>islessgreater()</i>	m	m	m	m	.	.	.	m	.
<i>islower()</i>	m	m	m	m	m	m	m	m	m
<i>islower_I()</i>	m	m
<i>isnan()</i>	m	m	m	m	m	m	.	.	.
<i>isnormal()</i>	m	m	m	m	.	.	.	m	.
<i>isprint()</i>	m	m	m	m	m	m	m	m	m
<i>isprint_I()</i>	m	m
<i>ispunct()</i>	m	m	m	m	m	m	m	m	m
<i>ispunct_I()</i>	m	m
<i>isspace()</i>	m	m	m	m	m	m	m	m	m
<i>isspace_I()</i>	m	m
<i>isunordered()</i>	m	m	m	m	.	.	.	m	.
<i>isupper()</i>	m	m	m	m	m	m	m	m	m

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>isupper_I()</i>	m	m
<i>iswalnum()</i>	m	m	m	m	m	m	.	m	1
<i>iswalnum_I()</i>	m	m
<i>iswalpha()</i>	m	m	m	m	m	m	.	m	1
<i>iswalpha_I()</i>	m	m
<i>iswblank()</i>	m	m	m	m	.	.	.	m	.
<i>iswblank_I()</i>	m	m
<i>iswcntrl()</i>	m	m	m	m	m	m	.	m	1
<i>iswcntrl_I()</i>	m	m
<i>iswctype()</i>	m	m	m	m	m	m	.	m	1
<i>iswctype_I()</i>	m	m
<i>iswdigit()</i>	m	m	m	m	m	m	.	m	1
<i>iswdigit_I()</i>	m	m
<i>iswgraph()</i>	m	m	m	m	m	m	.	m	1
<i>iswgraph_I()</i>	m	m
<i>iswlower()</i>	m	m	m	m	m	m	.	m	1
<i>iswlower_I()</i>	m	m
<i>iswprint()</i>	m	m	m	m	m	m	.	m	1
<i>iswprint_I()</i>	m	m
<i>iswpunct()</i>	m	m	m	m	m	m	.	m	1
<i>iswpunct_I()</i>	m	m
<i>iswspace()</i>	m	m	m	m	m	m	.	m	1
<i>iswspace_I()</i>	m	m
<i>iswupper()</i>	m	m	m	m	m	m	.	m	1
<i>iswupper_I()</i>	m	m
<i>iswxdigit()</i>	m	m	m	m	m	m	.	m	1
<i>iswxdigit_I()</i>	m	m
<i>isxdigit()</i>	m	m	m	m	m	m	m	m	m
<i>isxdigit_I()</i>	m	m
<i>j0()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>j1()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>jn()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>jrands48()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>kill()</i>	m	m	m	m	m	m	m	.	.
<i>killpg()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>l64a()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>labs()</i>	m	m	m	m	m	m	.	m	m
<i>lchown()</i>	m	m	m	xsi	m	m	.	.	.
<i>lcong48()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>ldexp()</i>	m	m	m	m	m	m	m	m	m
<i>ldexpf()</i>	m	m	m	m	.	.	.	m	.
<i>ldexpl()</i>	m	m	m	m	.	.	.	m	.
<i>ldiv()</i>	m	m	m	m	m	m	.	m	m
<i>lfind()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>lgamma()</i>	m	m	m	m	m	m	.	m	.
<i>lgammaf()</i>	m	m	m	m	.	.	.	m	.
<i>lgammal()</i>	m	m	m	m	.	.	.	m	.
<i>link()</i>	m	m	m	m	m	m	m	.	.
<i>linkat()</i>	m	m
<i>lio_listio()</i>	m	m	o	aio	o	.	r	.	.

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>listen()</i>	m	m	m	m	m	m	.	.	.
<i>llabs()</i>	m	m	m	m	.	.	.	m	.
<i>lldiv()</i>	m	m	m	m	.	.	.	m	.
<i>llrint()</i>	m	m	m	m	.	.	.	m	.
<i>llrintf()</i>	m	m	m	m	.	.	.	m	.
<i>llrintl()</i>	m	m	m	m	.	.	.	m	.
<i>llround()</i>	m	m	m	m	.	.	.	m	.
<i>llroundf()</i>	m	m	m	m	.	.	.	m	.
<i>llroundl()</i>	m	m	m	m	.	.	.	m	.
<i>localeconv()</i>	m	m	m	m	m	m	.	m	m
<i>localtime()</i>	m	m	m	m	m	m	m	m	m
<i>localtime_r()</i>	m	m	m	tsf	m	.	t	.	.
<i>lockf()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>log10()</i>	m	m	m	m	m	m	m	m	m
<i>log10f()</i>	m	m	m	m	.	.	.	m	.
<i>log10l()</i>	m	m	m	m	.	.	.	m	.
<i>log1p()</i>	m	m	m	m	m	m	.	m	.
<i>log1pf()</i>	m	m	m	m	.	.	.	m	.
<i>log1pl()</i>	m	m	m	m	.	.	.	m	.
<i>log2()</i>	m	m	m	m	.	.	.	m	.
<i>log2f()</i>	m	m	m	m	.	.	.	m	.
<i>log2l()</i>	m	m	m	m	.	.	.	m	.
<i>log()</i>	m	m	m	m	m	m	m	m	m
<i>logb()</i>	m	m	m	m	m	m	.	m	.
<i>logbf()</i>	m	m	m	m	.	.	.	m	.
<i>logbl()</i>	m	m	m	m	.	.	.	m	.
<i>logf()</i>	m	m	m	m	.	.	.	m	.
<i>logl()</i>	m	m	m	m	.	.	.	m	.
<i>longjmp()</i>	m	m	m	m	m	m	m	m	m
<i>lrand48()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>lrint()</i>	m	m	m	m	.	.	.	m	.
<i>lrintf()</i>	m	m	m	m	.	.	.	m	.
<i>lrintl()</i>	m	m	m	m	.	.	.	m	.
<i>lround()</i>	m	m	m	m	.	.	.	m	.
<i>lroundf()</i>	m	m	m	m	.	.	.	m	.
<i>lroundl()</i>	m	m	m	m	.	.	.	m	.
<i>lsearch()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>lseek()</i>	m	m	m	m	m	m	m	.	.
<i>lstat()</i>	m	m	m	m	m	m	.	.	.
<i>malloc()</i>	m	m	m	m	m	m	m	m	m
<i>mblen()</i>	m	m	m	m	m	m	.	m	m
<i>mbrlen()</i>	m	m	m	m	m	.	.	m	1
<i>mbrtowc()</i>	m	m	m	m	m	.	.	m	1
<i>mbsinit()</i>	m	m	m	m	m	.	.	m	1
<i>mbsnrtowcs()</i>	m	m
<i>mbsrtowcs()</i>	m	m	m	m	m	.	.	m	1
<i>mbstowcs()</i>	m	m	m	m	m	m	.	m	m
<i>mbtowc()</i>	m	m	m	m	m	m	.	m	m
<i>memccpy()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>memchr()</i>	m	m	m	m	m	m	.	m	m

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>memcpy()</i>	m	m	m	m	m	m	.	m	m
<i>memcopy()</i>	m	m	m	m	m	m	.	m	m
<i>memmove()</i>	m	m	m	m	m	m	.	m	m
<i>memset()</i>	m	m	m	m	m	m	.	m	m
<i>mkdir()</i>	m	m	m	m	m	m	m	.	.
<i>mkdirat()</i>	m	m
<i>mkdtemp()</i>	m	m
<i>mkfifo()</i>	m	m	m	m	m	m	m	.	.
<i>mkfifoat()</i>	m	m
<i>mknod()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>mknodat()</i>	m	xsi
<i>mkstemp()</i>	m	m	m	xsi	m	m	.	.	.
<i>mktime()</i>	m	m	m	m	m	m	m	m	m
<i>mlock()</i>	o	mlr	o	mr	o	.	r	.	.
<i>mlockall()</i>	o	ml	o	ml	o	.	r	.	.
<i>mmap()</i>	m	m	m	mf shm tym	m	m	.	.	.
<i>modf()</i>	m	m	m	m	m	m	m	m	m
<i>modff()</i>	m	m	m	m	.	.	.	m	.
<i>modfl()</i>	m	m	m	m	.	.	.	m	.
<i>mprotect()</i>	m	m	m	mpr	m	m	.	.	.
<i>mq_close()</i>	o	msg	o	msg	o	.	r	.	.
<i>mq_getattr()</i>	o	msg	o	msg	o	.	r	.	.
<i>mq_notify()</i>	o	msg	o	msg	o	.	r	.	.
<i>mq_open()</i>	o	msg	o	msg	o	.	r	.	.
<i>mq_receive()</i>	o	msg	o	msg	o	.	r	.	.
<i>mq_send()</i>	o	msg	o	msg	o	.	r	.	.
<i>mq_setattr()</i>	o	msg	o	msg	o	.	r	.	.
<i>mq_timedreceive()</i>	o	msg	o	msg
<i>mq_timedsend()</i>	o	msg	o	msg
<i>mq_unlink()</i>	o	msg	o	msg	o	.	r	.	.
<i>rand48()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>msgctl()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>msgget()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>msgrcv()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>msgsnd()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>msync()</i>	m	xsi sio	m	mf sio	m	m	.	.	.
<i>munlock()</i>	o	mlr	o	mr	o	.	r	.	.
<i>munlockall()</i>	o	ml	o	ml	o	.	r	.	.
<i>munmap()</i>	m	m	m	mf shm tym	m	m	.	.	.
<i>nan()</i>	m	m	m	m	.	.	.	m	.
<i>nanf()</i>	m	m	m	m	.	.	.	m	.
<i>nanl()</i>	m	m	m	m	.	.	.	m	.
<i>nanosleep()</i>	m	m	o	tmr	o	.	r	.	.
<i>nearbyint()</i>	m	m	m	m	.	.	.	m	.
<i>nearbyintf()</i>	m	m	m	m	.	.	.	m	.
<i>nearbyintl()</i>	m	m	m	m	.	.	.	m	.
<i>newlocale()</i>	m	m
<i>nextafter()</i>	m	m	m	m	m	m	.	m	.
<i>nextafterf()</i>	m	m	m	m	.	.	.	m	.
<i>nextafterl()</i>	m	m	m	m	.	.	.	m	.

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>nexttoward()</i>	m	m	m	m	.	.	.	m	.
<i>nexttowardf()</i>	m	m	m	m	.	.	.	m	.
<i>nexttowardl()</i>	m	m	m	m	.	.	.	m	.
<i>nftw()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>nice()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>nl_langinfo()</i>	m	m	m	xsi	m	m	.	.	.
<i>nl_langinfo_l()</i>	m	m
<i>nrnd48()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>ntohl()</i>	m	m	m	m	m	m	.	.	.
<i>ntohs()</i>	m	m	m	m	m	m	.	.	.
<i>open()</i>	m	m	m	m	m	m	m	.	.
<i>open_memstream()</i>	m	m
<i>open_wmemstream()</i>	m	m
<i>openat()</i>	m	m
<i>opendir()</i>	m	m	m	m	m	m	m	.	.
<i>openlog()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>optarg</i>	m	m	m	m	m	m	m	.	.
<i>opterr</i>	m	m	m	m	m	m	m	.	.
<i>optind</i>	m	m	m	m	m	m	m	.	.
<i>optopt</i>	m	m	m	m	m	m	m	.	.
<i>pathconf()</i>	m	m	m	m	m	m	m	.	.
<i>pause()</i>	m	m	m	m	m	m	m	.	.
<i>pclose()</i>	m	m	m	m	m	m	m	.	.
<i>perror()</i>	m	m	m	m	m	m	m	m	m
<i>pipe()</i>	m	m	m	m	m	m	m	.	.
<i>poll()</i>	m	m	m	xsi	m	m	.	.	.
<i>popen()</i>	m	m	m	m	m	m	m	.	.
<i>posix_fadvise()</i>	o	adv	o	adv
<i>posix_fallocate()</i>	o	adv	o	adv
<i>posix_madvise()</i>	o	adv	o	adv
<i>posix_mem_offset()</i>	o	tym	o	tym
<i>posix_memalign()</i>	o	adv	o	adv
<i>posix_openpt()</i>	m	xsi	m	xsi
<i>posix_spawn()</i>	o	spn	o	spn
<i>posix_spawn_file_actions_addclose()</i>	o	spn	o	spn
<i>posix_spawn_file_actions_adddup2()</i>	o	spn	o	spn
<i>posix_spawn_file_actions_addopen()</i>	o	spn	o	spn
<i>posix_spawn_file_actions_destroy()</i>	o	spn	o	spn
<i>posix_spawn_file_actions_init()</i>	o	spn	o	spn
<i>posix_spawnattr_destroy()</i>	o	spn	o	spn
<i>posix_spawnattr_getflags()</i>	o	spn	o	spn
<i>posix_spawnattr_getpgroup()</i>	o	spn	o	spn
<i>posix_spawnattr_getschedparam()</i>	o	spn ps	o	spn ps
<i>posix_spawnattr_getschedpolicy()</i>	o	spn ps	o	spn ps
<i>posix_spawnattr_getsigdefault()</i>	o	spn	o	spn
<i>posix_spawnattr_getsigmask()</i>	o	spn	o	spn
<i>posix_spawnattr_init()</i>	o	spn	o	spn
<i>posix_spawnattr_setflags()</i>	o	spn	o	spn
<i>posix_spawnattr_setpgroup()</i>	o	spn	o	spn
<i>posix_spawnattr_setschedparam()</i>	o	spn ps	o	spn ps

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>posix_spawnattr_setschedpolicy()</i>	o	spn ps	o	spn ps
<i>posix_spawnattr_setsigdefault()</i>	o	spn	o	spn
<i>posix_spawnattr_setsigmask()</i>	o	spn	o	spn
<i>posix_spawnnp()</i>	o	spn	o	spn
<i>posix_trace_attr_destroy()</i>	ob o	ob trc	o	trc
<i>posix_trace_attr_getclockres()</i>	ob o	ob trc	o	trc
<i>posix_trace_attr_getcreatetime()</i>	ob o	ob trc	o	trc
<i>posix_trace_attr_getgenversion()</i>	ob o	ob trc	o	trc
<i>posix_trace_attr_getinherited()</i>	o	tri	o	trc tri
<i>posix_trace_attr_getlogfullpolicy()</i>	o	trl	o	trc trl
<i>posix_trace_attr_getlogsize()</i>	o	trl	o	trc trl
<i>posix_trace_attr_getmaxdatasize()</i>	m	m	o	trc
<i>posix_trace_attr_getmaxsystemeventsz()</i>	m	m	o	trc
<i>posix_trace_attr_getmaxusereventsiz()</i>	m	m	o	trc
<i>posix_trace_attr_getname()</i>	ob o	ob trc	o	trc
<i>posix_trace_attr_getstreamfullpolicy()</i>	ob o	ob trc	o	trc
<i>posix_trace_attr_getstreamsize()</i>	ob o	ob trc	o	trc
<i>posix_trace_attr_init()</i>	ob o	ob trc	o	trc
<i>posix_trace_attr_setinherited()</i>	o	tri	o	trc tri
<i>posix_trace_attr_setlogfullpolicy()</i>	o	trl	o	trc trl
<i>posix_trace_attr_setlogsize()</i>	o	trl	o	trc trl
<i>posix_trace_attr_setmaxdatasize()</i>	ob o	ob trc	o	trc
<i>posix_trace_attr_setname()</i>	ob o	ob trc	o	trc
<i>posix_trace_attr_setstreamfullpolicy()</i>	ob o	ob trc	o	trc
<i>posix_trace_attr_setstreamsize()</i>	ob o	ob trc	o	trc
<i>posix_trace_clear()</i>	ob o	ob trc	o	trc
<i>posix_trace_close()</i>	o	trl	o	trc trl
<i>posix_trace_create()</i>	ob o	ob trc	o	trc
<i>posix_trace_create_withlog()</i>	o	trl	o	trc trl
<i>posix_trace_event()</i>	ob o	ob trc	o	trc
<i>posix_trace_eventid_equal()</i>	ob o	ob trc	o	trc
<i>posix_trace_eventid_get_name()</i>	ob o	ob trc	o	trc
<i>posix_trace_eventid_open()</i>	ob o	ob trc	o	trc
<i>posix_trace_eventset_add()</i>	o	tef	o	trc tef
<i>posix_trace_eventset_del()</i>	o	tef	o	trc tef
<i>posix_trace_eventset_empty()</i>	o	tef	o	trc tef
<i>posix_trace_eventset_fill()</i>	o	tef	o	trc tef
<i>posix_trace_eventset_ismember()</i>	o	tef	o	trc tef
<i>posix_trace_eventtypelist_getnext_id()</i>	ob o	ob trc	o	trc
<i>posix_trace_eventtypelist_rewind()</i>	ob o	ob trc	o	trc
<i>posix_trace_flush()</i>	o	trl	o	trc trl
<i>posix_trace_get_attr()</i>	ob o	ob trc	o	trc
<i>posix_trace_get_filter()</i>	o	tef	o	trc tef
<i>posix_trace_get_status()</i>	ob o	ob trc	o	trc
<i>posix_trace_getnext_event()</i>	ob o	ob trc	o	trc
<i>posix_trace_open()</i>	o	trl	o	trc trl
<i>posix_trace_rewind()</i>	o	trl	o	trc trl
<i>posix_trace_set_filter()</i>	o	tef	o	trc tef
<i>posix_trace_shutdown()</i>	ob o	ob trc	o	trc
<i>posix_trace_start()</i>	ob o	ob trc	o	trc

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>posix_trace_stop()</i>	ob o	ob trc	o	trc
<i>posix_trace_timedgetnext_event()</i>	ob o	ob trc	o	trc tmo
<i>posix_trace_trid_eventid_open()</i>	o	tef	o	trc tef
<i>posix_trace_trygetnext_event()</i>	ob o	ob trc	o	trc
<i>posix_typed_mem_get_info()</i>	o	tym	o	tym
<i>posix_typed_mem_open()</i>	o	tym	o	tym
<i>pow()</i>	m	m	m	m	m	m	m	m	m
<i>powf()</i>	m	m	m	m	.	.	.	m	.
<i>powl()</i>	m	m	m	m	.	.	.	m	.
<i>pread()</i>	m	m	m	xsi	m
<i>printf()</i>	m	m	m	m	m	m	m	m	m
<i>pselect()</i>	m	m	m	m
<i>psiginfo()</i>	m	m
<i>psignal()</i>	m	m
<i>pthread_atfork()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_attr_destroy()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_attr_getdetachstate()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_attr_getguardsize()</i>	m	m	m	xsi	m
<i>pthread_attr_getinheritsched()</i>	o	tps	o	thr tps	o	.	t	.	.
<i>pthread_attr_getschedparam()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_attr_getschedpolicy()</i>	o	tps	o	thr tps	o	.	t	.	.
<i>pthread_attr_getscope()</i>	o	tps	o	thr tps	o	.	t	.	.
<i>pthread_attr_getstack()</i>	m	tss	m	thr tss
<i>pthread_attr_getstacksize()</i>	m	tss	m	thr tss	m	.	t	.	.
<i>pthread_attr_init()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_attr_setdetachstate()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_attr_setguardsize()</i>	m	m	m	xsi	m
<i>pthread_attr_setinheritsched()</i>	o	tps	o	thr tps	o	.	t	.	.
<i>pthread_attr_setschedparam()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_attr_setschedpolicy()</i>	o	tps	o	thr tps	o	.	t	.	.
<i>pthread_attr_setscope()</i>	o	tps	o	thr tps	o	.	t	.	.
<i>pthread_attr_setstack()</i>	m	tss	m	thr tss
<i>pthread_attr_setstacksize()</i>	m	tss	m	thr tss	m	.	t	.	.
<i>pthread_barrier_destroy()</i>	m	m	o	thr bar
<i>pthread_barrier_init()</i>	m	m	o	thr bar
<i>pthread_barrier_wait()</i>	m	m	o	thr bar
<i>pthread_barrierattr_destroy()</i>	m	m	o	thr bar
<i>pthread_barrierattr_getpshared()</i>	m	tsh	o	thr bar tsh
<i>pthread_barrierattr_init()</i>	m	m	o	thr bar
<i>pthread_barrierattr_setpshared()</i>	m	tsh	o	thr bar tsh
<i>pthread_cancel()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_cleanup_pop()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_cleanup_push()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_cond_broadcast()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_cond_destroy()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_cond_init()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_cond_signal()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_cond_timedwait()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_cond_wait()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_condattr_destroy()</i>	m	m	m	thr	m	.	t	.	.

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>pthread_condattr_getclock()</i>	m	m	o	thr cs
<i>pthread_condattr_getpshared()</i>	m	tsh	m	thr tsh	m	.	t	.	.
<i>pthread_condattr_init()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_condattr_setclock()</i>	m	m	o	thr cs
<i>pthread_condattr_setpshared()</i>	m	tsh	m	thr tsh	m	.	t	.	.
<i>pthread_create()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_detach()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_equal()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_exit()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_getconcurrency()</i>	ob	ob xsi	m	xsi	m
<i>pthread_getcpuclockid()</i>	o	tct	o	thr tct
<i>pthread_getschedparam()</i>	o	tps	o	thr tps	o	.	t	.	.
<i>pthread_getspecific()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_join()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_key_create()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_key_delete()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_kill()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_mutex_consistent()</i>	m	m
<i>pthread_mutex_destroy()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_mutex_getprioceiling()</i>	o	rpp tpp	o	thr tpp	o	.	t	.	.
<i>pthread_mutex_init()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_mutex_lock()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_mutex_setprioceiling()</i>	o	rpp tpp	o	thr tpp	o	.	t	.	.
<i>pthread_mutex_timedlock()</i>	m	m	o	thr tmo
<i>pthread_mutex_trylock()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_mutex_unlock()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_mutexattr_destroy()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_mutexattr_getprioceiling()</i>	o	rpp tpp	o	thr tpp	o	.	t	.	.
<i>pthread_mutexattr_getprotocol()</i>	o	mc1	o	thr tpp tpi	o	.	t	.	.
<i>pthread_mutexattr_getpshared()</i>	m	tsh	m	thr tsh	m	.	t	.	.
<i>pthread_mutexattr_getrobust()</i>	m	m
<i>pthread_mutexattr_gettype()</i>	m	m	m	xsi	m
<i>pthread_mutexattr_init()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_mutexattr_setprioceiling()</i>	o	rpp tpp	o	thr tpp	o	.	t	.	.
<i>pthread_mutexattr_setprotocol()</i>	o	mc1	o	thr tpp tpi	o	.	t	.	.
<i>pthread_mutexattr_setpshared()</i>	m	tsh	m	thr tsh	m	.	t	.	.
<i>pthread_mutexattr_setrobust()</i>	m	m
<i>pthread_mutexattr_settype()</i>	m	m	m	xsi	m
<i>pthread_once()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_rwlock_destroy()</i>	m	m	m	thr	m
<i>pthread_rwlock_init()</i>	m	m	m	thr	m
<i>pthread_rwlock_rdlock()</i>	m	m	m	thr	m
<i>pthread_rwlock_timedrdlock()</i>	m	m	o	thr tmo
<i>pthread_rwlock_timedwrlock()</i>	m	m	o	thr tmo
<i>pthread_rwlock_tryrdlock()</i>	m	m	m	thr	m
<i>pthread_rwlock_trywrlock()</i>	m	m	m	thr	m
<i>pthread_rwlock_unlock()</i>	m	m	m	thr	m
<i>pthread_rwlock_wrlock()</i>	m	m	m	thr	m
<i>pthread_rwlockattr_destroy()</i>	m	m	m	thr	m
<i>pthread_rwlockattr_getpshared()</i>	m	tsh	m	thr tsh	m

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>pthread_rwlockattr_init()</i>	m	m	m	thr	m
<i>pthread_rwlockattr_setpshared()</i>	m	tsh	m	thr tsh	m
<i>pthread_self()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_setcancelstate()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_setcanceltype()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_setconcurrency()</i>	ob	ob xsi	m	xsi	m
<i>pthread_setschedparam()</i>	o	tps	o	thr tps	o	.	t	.	.
<i>pthread_setschedprio()</i>	o	tps	o	thr tps
<i>pthread_setspecific()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_sigmask()</i>	m	m	m	thr	m	.	t	.	.
<i>pthread_spin_destroy()</i>	m	m	o	thr spi
<i>pthread_spin_init()</i>	m	m	o	thr spi
<i>pthread_spin_lock()</i>	m	m	o	thr spi
<i>pthread_spin_trylock()</i>	m	m	o	thr spi
<i>pthread_spin_unlock()</i>	m	m	o	thr spi
<i>pthread_testcancel()</i>	m	m	m	thr	m	.	t	.	.
<i>ptsname()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>putc()</i>	m	m	m	m	m	m	m	m	m
<i>putc_unlocked()</i>	m	m	m	tsf	m	.	t	.	.
<i>putchar()</i>	m	m	m	m	m	m	m	m	m
<i>putchar_unlocked()</i>	m	m	m	tsf	m	.	t	.	.
<i>putenv()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>putmsg()</i>	ob o	ob xsr	o	xsr	m	m	.	.	.
<i>putpmsg()</i>	ob o	ob xsr	o	xsr	m	m	.	.	.
<i>puts()</i>	m	m	m	m	m	m	m	m	m
<i>pututxline()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>putwc()</i>	m	m	m	m	m	m	.	m	1
<i>putwchar()</i>	m	m	m	m	m	m	.	m	1
<i>pwrite()</i>	m	m	m	xsi	m
<i>qsort()</i>	m	m	m	m	m	m	m	m	m
<i>raise()</i>	m	m	m	m	m	m	.	m	m
<i>rand()</i>	m	m	m	m	m	m	m	m	m
<i>rand_r()</i>	ob	ob	m	tsf	m	.	t	.	.
<i>random()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>read()</i>	m	m	m	xsi	m	m	m	.	.
<i>readdir()</i>	m	m	m	m	m	m	m	.	.
<i>readdir_r()</i>	m	m	m	tsf	m	.	t	.	.
<i>readlink()</i>	m	m	m	m	m	m	.	.	.
<i>readlinkat()</i>	m	m
<i>readv()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>realloc()</i>	m	m	m	m	m	m	m	m	m
<i>realpath()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>recv()</i>	m	m	m	m	m	m	.	.	.
<i>recvfrom()</i>	m	m	m	m	m	m	.	.	.
<i>recvmsg()</i>	m	m	m	m	m	m	.	.	.
<i>regcomp()</i>	m	m	m	m	m	o	m	.	.
<i>regerror()</i>	m	m	m	m	m	o	m	.	.
<i>regexexec()</i>	m	m	m	m	m	o	m	.	.
<i>regfree()</i>	m	m	m	m	m	o	m	.	.
<i>remainder()</i>	m	m	m	m	m	m	.	m	.

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>remainderf()</i>	m	m	m	m	.	.	.	m	.
<i>remainderl()</i>	m	m	m	m	.	.	.	m	.
<i>remove()</i>	m	m	m	m	m	m	m	m	m
<i>remque()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>remquo()</i>	m	m	m	m	.	.	.	m	.
<i>remquof()</i>	m	m	m	m	.	.	.	m	.
<i>remquol()</i>	m	m	m	m	.	.	.	m	.
<i>rename()</i>	m	m	m	m	m	m	m	m	m
<i>renameat()</i>	m	m
<i>rewind()</i>	m	m	m	m	m	m	m	m	m
<i>rewinddir()</i>	m	m	m	m	m	m	m	.	.
<i>rint()</i>	m	m	m	m	m	m	.	m	.
<i>rintf()</i>	m	m	m	m	.	.	.	m	.
<i>rintl()</i>	m	m	m	m	.	.	.	m	.
<i>rmdir()</i>	m	m	m	m	m	m	m	.	.
<i>round()</i>	m	m	m	m	.	.	.	m	.
<i>roundf()</i>	m	m	m	m	.	.	.	m	.
<i>roundl()</i>	m	m	m	m	.	.	.	m	.
<i>scalbln()</i>	m	m	m	m	.	.	.	m	.
<i>scalblnf()</i>	m	m	m	m	.	.	.	m	.
<i>scalblnl()</i>	m	m	m	m	.	.	.	m	.
<i>scalbn()</i>	m	m	m	m	.	.	.	m	.
<i>scalbnf()</i>	m	m	m	m	.	.	.	m	.
<i>scalbnl()</i>	m	m	m	m	.	.	.	m	.
<i>scandir()</i>	m	m
<i>scanf()</i>	m	m	m	m	m	m	m	m	m
<i>sched_get_priority_max()</i>	o	ps tps	o	ps	o	.	r	.	.
<i>sched_get_priority_min()</i>	o	ps tps	o	ps	o	.	r	.	.
<i>sched_getparam()</i>	o	ps	o	ps	o	.	r	.	.
<i>sched_getscheduler()</i>	o	ps	o	ps	o	.	r	.	.
<i>sched_rr_get_interval()</i>	o	ps tps	o	ps	o	.	r	.	.
<i>sched_setparam()</i>	o	ps	o	ps	o	.	r	.	.
<i>sched_setscheduler()</i>	o	ps	o	ps	o	.	r	.	.
<i>sched_yield()</i>	m	m	m	ps thr	m	.	r	.	.
<i>seed48()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>seekdir()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>select()</i>	m	m	m	m	m	m	.	.	.
<i>sem_close()</i>	m	m	o	sem	o	.	r	.	.
<i>sem_destroy()</i>	m	m	o	sem	o	.	r	.	.
<i>sem_getvalue()</i>	m	m	o	sem	o	.	r	.	.
<i>sem_init()</i>	m	m	o	sem	o	.	r	.	.
<i>sem_open()</i>	m	m	o	sem	o	.	r	.	.
<i>sem_post()</i>	m	m	o	sem	o	.	r	.	.
<i>sem_timedwait()</i>	m	m	o	sem tmo
<i>sem_trywait()</i>	m	m	o	sem	o	.	r	.	.
<i>sem_unlink()</i>	m	m	o	sem	o	.	r	.	.
<i>sem_wait()</i>	m	m	o	sem	o	.	r	.	.
<i>semctl()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>semget()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>semop()</i>	m	xsi	m	xsi	m	m	.	.	.

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>send()</i>	m	m	m	m	m	m	.	.	.
<i>sendmsg()</i>	m	m	m	m	m	m	.	.	.
<i>sendto()</i>	m	m	m	m	m	m	.	.	.
<i>setbuf()</i>	m	m	m	m	m	m	m	m	m
<i>setegid()</i>	m	m	m	m
<i>setenv()</i>	m	m	m	m
<i>seteuid()</i>	m	m	m	m
<i>setgid()</i>	m	m	m	m	m	m	m	.	.
<i>setgrent()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>sethostent()</i>	m	m	m	m	m	m	.	.	.
<i>setitimer()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>setjmp()</i>	m	m	m	m	m	m	m	m	m
<i>setkey()</i>	m	xsi	o	xsi	o	o	.	.	.
<i>setlocale()</i>	m	m	m	m	m	m	m	m	m
<i>setlogmask()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>setnetent()</i>	m	m	m	m	m	m	.	.	.
<i>setpgid()</i>	m	m	m	m	m	m	m	.	.
<i>setpgrp()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>setpriority()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>setprotoent()</i>	m	m	m	m	m	m	.	.	.
<i>setpwent()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>setregid()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>setreuid()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>setrlimit()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>setservent()</i>	m	m	m	m	m	m	.	.	.
<i>setsid()</i>	m	m	m	m	m	m	m	.	.
<i>setsockopt()</i>	m	m	m	m	m	m	.	.	.
<i>setstate()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>setuid()</i>	m	m	m	m	m	m	m	.	.
<i>setutxent()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>setvbuf()</i>	m	m	m	m	m	m	.	m	m
<i>shm_open()</i>	o	shm	o	shm	o	.	r	.	.
<i>shm_unlink()</i>	o	shm	o	shm	o	.	r	.	.
<i>shmat()</i>	m	xsi	m	xsi	m	o	.	.	.
<i>shmctl()</i>	m	xsi	m	xsi	m	o	.	.	.
<i>shmdt()</i>	m	xsi	m	xsi	m	o	.	.	.
<i>shmget()</i>	m	xsi	m	xsi	m	o	.	.	.
<i>shutdown()</i>	m	m	m	m	m	m	.	.	.
<i>sigaction()</i>	m	m	m	m	m	m	m	.	.
<i>sigaddset()</i>	m	m	m	m	m	m	m	.	.
<i>sigaltstack()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>sigdelset()</i>	m	m	m	m	m	m	m	.	.
<i>sigemptyset()</i>	m	m	m	m	m	m	m	.	.
<i>sigfillset()</i>	m	m	m	m	m	m	m	.	.
<i>sighold()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>sigignore()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>siginterrupt()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>sigismember()</i>	m	m	m	m	m	m	m	.	.
<i>siglongjmp()</i>	m	m	m	m	m	m	m	.	.
<i>signal()</i>	m	m	m	m	m	m	.	m	m

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>signbit()</i>	m	m	m	m	.	.	.	m	.
<i>signgam</i>	m	xsi	m	xsi
<i>sigpause()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>sigpending()</i>	m	m	m	m	m	m	m	.	.
<i>sigprocmask()</i>	m	m	m	thr	m	m	m	.	.
<i>sigqueue()</i>	m	m	o	rts	o	.	r	.	.
<i>sigrelse()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>sigset()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>sigsetjmp()</i>	m	m	m	m	m	m	m	.	.
<i>sigsuspend()</i>	m	m	m	m	m	m	m	.	.
<i>sigtimedwait()</i>	m	m	o	rts	o	.	r	.	.
<i>sigwait()</i>	m	m	m	m	m	.	t	.	.
<i>sigwaitinfo()</i>	m	m	o	rts	o	.	r	.	.
<i>sin()</i>	m	m	m	m	m	m	m	m	m
<i>sinf()</i>	m	m	m	m	.	.	.	m	.
<i>sinh()</i>	m	m	m	m	m	m	m	m	m
<i>sinhf()</i>	m	m	m	m	.	.	.	m	.
<i>sinhl()</i>	m	m	m	m	.	.	.	m	.
<i>sinl()</i>	m	m	m	m	.	.	.	m	.
<i>sleep()</i>	m	m	m	m	m	m	m	.	.
<i>snprintf()</i>	m	m	m	m	m	.	.	m	.
<i>socketatmark()</i>	m	m	m	m
<i>socket()</i>	m	m	m	m	m	m	.	.	.
<i>socketpair()</i>	m	m	m	m	m	m	.	.	.
<i>sprintf()</i>	m	m	m	m	m	m	m	m	m
<i>sqrt()</i>	m	m	m	m	m	m	m	m	m
<i>sqrtf()</i>	m	m	m	m	.	.	.	m	.
<i>sqrtl()</i>	m	m	m	m	.	.	.	m	.
<i>srand48()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>srand()</i>	m	m	m	m	m	m	m	m	m
<i>srandom()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>sscanf()</i>	m	m	m	m	m	m	m	m	m
<i>stat()</i>	m	m	m	m	m	m	m	.	.
<i>statvfs()</i>	m	m	m	xsi	m	m	.	.	.
<i>stderr</i>	m	m	m	m	m	m	m	m	m
<i>stdin</i>	m	m	m	m	m	m	m	m	m
<i>stdout</i>	m	m	m	m	m	m	m	m	m
<i>stpncpy()</i>	m	m
<i>stpncpy()</i>	m	m
<i>strcasecmp()</i>	m	m	m	xsi	m	m	.	.	.
<i>strcasecmp_l()</i>	m	m
<i>strcat()</i>	m	m	m	m	m	m	m	m	m
<i>strchr()</i>	m	m	m	m	m	m	m	m	m
<i>strcmp()</i>	m	m	m	m	m	m	m	m	m
<i>strcoll()</i>	m	m	m	m	m	m	.	m	m
<i>strcoll_l()</i>	m	m
<i>strcpy()</i>	m	m	m	m	m	m	m	m	m
<i>strcspn()</i>	m	m	m	m	m	m	m	m	m
<i>strdup()</i>	m	m	m	xsi	m	m	.	.	.
<i>strerror()</i>	m	m	m	m	m	m	.	m	m

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>strerror_l()</i>	m	m
<i>strerror_r()</i>	m	m	m	tsf
<i>strfmon()</i>	m	m	m	xsi	m	o	.	.	.
<i>strfmon_l()</i>	m	m
<i>strftime()</i>	m	m	m	m	m	m	m	m	m
<i>strftime_l()</i>	m	m
<i>strlen()</i>	m	m	m	m	m	m	m	m	m
<i>strncasecmp()</i>	m	m	m	xsi	m	m	.	.	.
<i>strncasecmp_l()</i>	m	m
<i>strncat()</i>	m	m	m	m	m	m	m	m	m
<i>strncmp()</i>	m	m	m	m	m	m	m	m	m
<i>strncpy()</i>	m	m	m	m	m	m	m	m	m
<i>strndup()</i>	m	m
<i>strnlen()</i>	m	m
<i>strpbrk()</i>	m	m	m	m	m	m	m	m	m
<i>strptime()</i>	m	xsi	m	xsi	m	o	.	.	.
<i>strrchr()</i>	m	m	m	m	m	m	m	m	m
<i>strsignal()</i>	m	m
<i>strspn()</i>	m	m	m	m	m	m	m	m	m
<i>strstr()</i>	m	m	m	m	m	m	m	m	m
<i>strtod()</i>	m	m	m	m	m	m	.	m	m
<i>strtof()</i>	m	m	m	m	.	.	.	m	.
<i>strtoimax()</i>	m	m	m	m	.	.	.	m	.
<i>strtok()</i>	m	m	m	m	m	m	m	m	m
<i>strtok_r()</i>	m	m	m	tsf	m	.	t	.	.
<i>strtol()</i>	m	m	m	m	m	m	.	m	m
<i>strtold()</i>	m	m	m	m	.	.	.	m	.
<i>strtoll()</i>	m	m	m	m	.	.	.	m	.
<i>strtoul()</i>	m	m	m	m	m	m	.	m	m
<i>strtoull()</i>	m	m	m	m	.	.	.	m	.
<i>strtoumax()</i>	m	m	m	m	.	.	.	m	.
<i>strxfrm()</i>	m	m	m	m	m	m	.	m	m
<i>strxfrm_l()</i>	m	m
<i>swab()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>swprintf()</i>	m	m	m	m	m	.	.	m	1
<i>swscanf()</i>	m	m	m	m	m	.	.	m	1
<i>symlink()</i>	m	m	m	m	m	m	.	.	.
<i>symlinkat()</i>	m	m
<i>sync()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>sysconf()</i>	m	m	m	m	m	m	m	.	.
<i>syslog()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>system()</i>	m	m	m	m	m	m	.	m	m
<i>tan()</i>	m	m	m	m	m	m	m	m	m
<i>tanf()</i>	m	m	m	m	.	.	.	m	.
<i>tanh()</i>	m	m	m	m	m	m	m	m	m
<i>tanhf()</i>	m	m	m	m	.	.	.	m	.
<i>tanh_l()</i>	m	m	m	m	.	.	.	m	.
<i>tanl()</i>	m	m	m	m	.	.	.	m	.
<i>tcdrain()</i>	m	m	m	m	m	m	m	.	.
<i>tcflow()</i>	m	m	m	m	m	m	m	.	.

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>tcflush()</i>	m	m	m	m	m	m	m	.	.
<i>tcgetattr()</i>	m	m	m	m	m	m	m	.	.
<i>tcgetpgrp()</i>	m	m	m	m	m	m	m	.	.
<i>tcgetsid()</i>	m	m	m	xsi	m	m	.	.	.
<i>tcsendbreak()</i>	m	m	m	m	m	m	m	.	.
<i>tcsetattr()</i>	m	m	m	m	m	m	m	.	.
<i>tcsetpgrp()</i>	m	m	m	m	m	m	m	.	.
<i>tdelete()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>telldir()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>tempnam()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>tfind()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>tgamma()</i>	m	m	m	m	.	.	.	m	.
<i>tgammaf()</i>	m	m	m	m	.	.	.	m	.
<i>tgamma1()</i>	m	m	m	m	.	.	.	m	.
<i>time()</i>	m	m	m	m	m	m	m	m	m
<i>timer_create()</i>	m	m	o	tmr	o	.	r	.	.
<i>timer_delete()</i>	m	m	o	tmr	o	.	r	.	.
<i>timer_getoverrun()</i>	m	m	o	tmr	o	.	r	.	.
<i>timer_gettime()</i>	m	m	o	tmr	o	.	r	.	.
<i>timer_settime()</i>	m	m	o	tmr	o	.	r	.	.
<i>times()</i>	m	m	m	m	m	m	m	.	.
<i>timezone</i>	m	xsi	m	xsi	m	m	.	.	.
<i>tmpfile()</i>	m	m	m	m	m	m	m	m	m
<i>tmpnam()</i>	ob	ob	m	m	m	m	m	m	m
<i>toascii()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>tolower()</i>	m	m	m	m	m	m	m	m	m
<i>tolower_l()</i>	m	m
<i>toupper()</i>	m	m	m	m	m	m	m	m	m
<i>toupper_l()</i>	m	m
<i>towctrans()</i>	m	m	m	m	m	.	.	m	1
<i>towctrans_l()</i>	m	m
<i>towlower()</i>	m	m	m	m	m	m	.	m	1
<i>towlower_l()</i>	m	m
<i>towupper()</i>	m	m	m	m	m	m	.	m	1
<i>towupper_l()</i>	m	m
<i>trunc()</i>	m	m	m	m	.	.	.	m	.
<i>truncate()</i>	m	m	m	xsi	m	m	.	.	.
<i>truncf()</i>	m	m	m	m	.	.	.	m	.
<i>truncl()</i>	m	m	m	m	.	.	.	m	.
<i>tsearch()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>ttynam()</i>	m	m	m	tsf	m	m	m	.	.
<i>ttynam_r()</i>	m	m	m	tsf	m	.	t	.	.
<i>twalk()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>tzname</i>	m	m	m	xsi	m	m	m	.	.
<i>tzset()</i>	m	m	m	xsi	m	m	m	.	.
<i>ulimit()</i>	ob	ob xsi	m	xsi	m	m	.	.	.
<i>umask()</i>	m	m	m	m	m	m	m	.	.
<i>uname()</i>	m	m	m	m	m	m	m	.	.
<i>ungetc()</i>	m	m	m	m	m	m	m	m	m
<i>ungetwc()</i>	m	m	m	m	m	m	.	m	1

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>unlink()</i>	m	m	m	m	m	m	m	.	.
<i>unlinkat()</i>	m	m
<i>unlockpt()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>unsetenv()</i>	m	m	m	m
<i>uselocale()</i>	m	m
<i>utime()</i>	ob	ob	m	m	m	m	m	.	.
<i>utimensat()</i>	m	m
<i>utimes()</i>	m	xsi	l	xsi	m	m	.	.	.
<i>va_arg()</i>	m	m	m	m
<i>va_copy()</i>	m	m	m	m
<i>va_end()</i>	m	m	m	m
<i>va_start()</i>	m	m	m	m
<i>vdprintf()</i>	m	m
<i>vfprintf()</i>	m	m	m	m	m	m	.	.	.
<i>vfprintf()</i>	m	m	m	m	.	.	.	m	.
<i>vfwprintf()</i>	m	m	m	m	m	.	.	m	1
<i>vfwscanf()</i>	m	m	m	m	.	.	.	m	.
<i>vprintf()</i>	m	m	m	m	m	m	.	m	m
<i>vscanf()</i>	m	m	m	m	.	.	.	m	.
<i>vsnprintf()</i>	m	m	m	m	m	.	.	m	.
<i>vsprintf()</i>	m	m	m	m	m	m	.	m	m
<i>vsscanf()</i>	m	m	m	m	.	.	.	m	.
<i>vswprintf()</i>	m	m	m	m	m	.	.	m	1
<i>vswscanf()</i>	m	m	m	m	.	.	.	m	.
<i>vwprintf()</i>	m	m	m	m	m	.	.	m	1
<i>vwscanf()</i>	m	m	m	m	.	.	.	m	.
<i>wait()</i>	m	m	m	m	m	m	m	.	.
<i>waitid()</i>	m	m	m	xsi	m	m	.	.	.
<i>waitpid()</i>	m	m	m	m	m	m	m	.	.
<i>wcpcpy()</i>	m	m
<i>wcpncpy()</i>	m	m
<i>wcrtomb()</i>	m	m	m	m	m	.	.	m	1
<i>wcscasecmp()</i>	m	m
<i>wcscasecmp_l()</i>	m	m
<i>wcscat()</i>	m	m	m	m	m	m	.	m	1
<i>wcschr()</i>	m	m	m	m	m	m	.	m	1
<i>wcscmp()</i>	m	m	m	m	m	m	.	m	1
<i>wcscoll()</i>	m	m	m	m	m	o	.	m	1
<i>wcscoll_l()</i>	m	m
<i>wcscpy()</i>	m	m	m	m	m	m	.	m	1
<i>wcscspn()</i>	m	m	m	m	m	m	.	m	1
<i>wcsdup()</i>	m	m
<i>wcsftime()</i>	m	m	m	m	m	o	.	m	1
<i>wcslen()</i>	m	m	m	m	m	m	.	m	1
<i>wcsncasecmp()</i>	m	m
<i>wcsncasecmp_l()</i>	m	m
<i>wcsncat()</i>	m	m	m	m	m	m	.	m	1
<i>wcsncmp()</i>	m	m	m	m	m	m	.	m	1
<i>wcsncpy()</i>	m	m	m	m	m	m	.	m	1
<i>wcsnlen()</i>	m	m

Interface	XSI	POSIX	U03	P01	U98	U95	P96 P92	C99	C89
<i>wcsnrtombs()</i>	m	m
<i>wcspbrk()</i>	m	m	m	m	m	m	.	m	1
<i>wcsrchr()</i>	m	m	m	m	m	m	.	m	1
<i>wcsrtombs()</i>	m	m	m	m	m	.	.	m	1
<i>wcsspn()</i>	m	m	m	m	m	m	.	m	1
<i>wcsstr()</i>	m	m	m	m	m	.	.	m	1
<i>wcstod()</i>	m	m	m	m	m	m	.	m	1
<i>wcstof()</i>	m	m	m	m	.	.	.	m	.
<i>wcstoimax()</i>	m	m	m	m	.	.	.	m	.
<i>wcstok()</i>	m	m	m	m	m	m	.	m	1
<i>wcstol()</i>	m	m	m	m	m	m	.	m	1
<i>wcstold()</i>	m	m	m	m	.	.	.	m	.
<i>wcstoll()</i>	m	m	m	m	.	.	.	m	.
<i>wcstombs()</i>	m	m	m	m	m	m	.	m	m
<i>wcstoul()</i>	m	m	m	m	m	m	.	m	1
<i>wcstoull()</i>	m	m	m	m	.	.	.	m	.
<i>wcstoumax()</i>	m	m	m	m	.	.	.	m	.
<i>wcswidth()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>wcsxfrm()</i>	m	m	m	m	m	o	.	m	1
<i>wcsxfrm_l()</i>	m	m
<i>wctob()</i>	m	m	m	m	m	.	.	m	1
<i>wctomb()</i>	m	m	m	m	m	m	.	m	m
<i>wctrans()</i>	m	m	m	m	m	.	.	m	1
<i>wctrans_l()</i>	m	m
<i>wctype()</i>	m	m	m	m	m	m	.	m	1
<i>wctype_l()</i>	m	m
<i>wcwidth()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>wmemchr()</i>	m	m	m	m	m	.	.	m	1
<i>wmemcmp()</i>	m	m	m	m	m	.	.	m	1
<i>wmemcpy()</i>	m	m	m	m	m	.	.	m	1
<i>wmemmove()</i>	m	m	m	m	m	.	.	m	1
<i>wmemset()</i>	m	m	m	m	m	.	.	m	1
<i>wordexp()</i>	m	m	m	m	m	o	m	.	.
<i>wordfree()</i>	m	m	m	m	m	o	m	.	.
<i>wprintf()</i>	m	m	m	m	m	.	.	m	1
<i>write()</i>	m	m	m	xsi	m	m	m	.	.
<i>writev()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>wscanf()</i>	m	m	m	m	m	.	.	m	1
<i>y0()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>y1()</i>	m	xsi	m	xsi	m	m	.	.	.
<i>yn()</i>	m	xsi	m	xsi	m	m	.	.	.

Utility Interface Table

8.1 Introduction

This chapter lists all the utilities described in XCU, Issue 7, complete with an indication of their status for XSI-conforming and POSIX-conforming systems, and indicates what other specifications support the interfaces.

The following conventions are used in columns 2 through 8:

- m** Indicates that the interface is defined as mandatory.
- d** Indicates that the interface is part of the DEVELOPMENT Option or Feature Group.
- f** Indicates that the interface is part of the FORTRAN Option or Feature Group.
- 2d** Indicates that the interface is part of IEEE Std 1003.2d-1994 (Batch Environment).
- o** Indicates that the interface is optional.
- ob** Indicates that the interface is Obsolescent, and although mandatory for the implementation, applications are discouraged from its use.
- ob o** In the XSI column, indicates that the interface is Obsolescent, and is optional. Applications are discouraged from its use.
- opt* In the POSIX and POSIX 01 columns, two or three letter option codes are used as described in [Section 1.4](#) (on page 2), denoting the option to which the interface belongs.
- ob opt** Same as *opt*, but the interface is also obsolescent.
- .** Indicates that the interface is not specified.

It should be noted that while another specification may support the interface, some of the interface semantics may have changed with evolution and standardization. A developer should not assume that because the interface appears in other specifications, it will behave exactly as described in the Single UNIX Specification, Version 4.

8.2 Utility Interface Table

There are 160 utilities listed.

Interface	XSI	POSIX	UNIX 03	POSIX 01	UNIX 98	UNIX 95	POSIX.2-92
<i>admin</i>	d	xsi	d	xsi	d	d	.
<i>alias</i>	m	m	m	up	m	m	o
<i>ar</i>	m	m	m	sd	m	m	o
<i>asa</i>	o	fr	o	fr	m	m	o
<i>at</i>	m	m	m	up	m	m	o
<i>awk</i>	m	m	m	m	m	m	m
<i>basename</i>	m	m	m	m	m	m	m
<i>batch</i>	m	m	m	up	m	m	o
<i>bc</i>	m	m	m	m	m	m	m
<i>bg</i>	m	up	m	up	m	m	o
<i>c99</i>	m	cd	m	cd	.	.	.
<i>cal</i>	m	xsi	m	xsi	m	m	.
<i>cat</i>	m	m	m	m	m	m	m
<i>cd</i>	m	m	m	m	m	m	m
<i>cflow</i>	d	xsi	d	xsi	d	d	.
<i>chgrp</i>	m	m	m	m	m	m	m
<i>chmod</i>	m	m	m	m	m	m	m
<i>chown</i>	m	m	m	m	m	m	m
<i>cksum</i>	m	m	m	m	m	m	m
<i>cmp</i>	m	m	m	m	m	m	m
<i>comm</i>	m	m	m	m	m	m	m
<i>command</i>	m	m	m	up	m	m	m
<i>compress</i>	m	xsi	m	xsi	m	m	.
<i>cp</i>	m	m	m	m	m	m	m
<i>crontab</i>	m	m	m	up	m	m	o
<i>csplit</i>	m	m	m	up	m	m	o
<i>ctags</i>	df	sd	df	sd up	df	df	o
<i>cut</i>	m	m	m	m	m	m	m
<i>cxref</i>	d	xsi	d	xsi	d	d	.
<i>date</i>	m	m	m	m	m	m	m
<i>dd</i>	m	m	m	m	m	m	m
<i>delta</i>	d	xsi	d	xsi	d	d	.
<i>df</i>	m	m	m	up	m	m	o
<i>diff</i>	m	m	m	m	m	m	m
<i>dirname</i>	m	m	m	m	m	m	m
<i>du</i>	m	m	m	up	m	m	o
<i>echo</i>	m	m	m	m	m	m	m
<i>ed</i>	m	m	m	m	m	m	m
<i>env</i>	m	m	m	m	m	m	m
<i>ex</i>	m	up	m	up	m	m	o
<i>expand</i>	m	m	m	up	m	m	o
<i>expr</i>	m	m	m	m	m	m	m
<i>false</i>	m	m	m	m	m	m	m
<i>fc</i>	m	up	m	up	m	m	o
<i>fg</i>	m	up	m	up	m	m	o
<i>file</i>	m	m	m	up	m	m	o
<i>find</i>	m	m	m	m	m	m	m
<i>fold</i>	m	m	m	m	m	m	m
<i>fort77</i>	f	fd	f	fd	f	f	o
<i>fuser</i>	m	xsi	m	xsi	m	.	.
<i>gencat</i>	m	m	m	xsi	m	m	.

Interface	XSI	POSIX	UNIX 03	POSIX 01	UNIX 98	UNIX 95	POSIX.2-92
<i>get</i>	d	xsi	d	xsi	d	d	.
<i>getconf</i>	m	m	m	m	m	m	.
<i>getopts</i>	m	m	m	m	m	m	.
<i>grep</i>	m	m	m	m	m	m	m
<i>hash</i>	m	m	m	xsi	m	m	.
<i>head</i>	m	m	m	m	m	m	m
<i>iconv</i>	m	m	m	m	m	m	.
<i>id</i>	m	m	m	m	m	m	m
<i>ipcrm</i>	m	xsi	m	xsi	m	.	.
<i>ipcs</i>	m	xsi	m	xsi	m	.	.
<i>jobs</i>	m	up	m	up	m	m	o
<i>join</i>	m	m	m	m	m	m	m
<i>kill</i>	m	m	m	m	m	m	m
<i>lex</i>	d	cd	d	cd	d	d	o
<i>link</i>	m	xsi	m	xsi	m	.	.
<i>ln</i>	m	m	m	m	m	m	m
<i>locale</i>	m	m	m	m	m	m	m
<i>localedef</i>	m	m	m	m	m	m	m
<i>logger</i>	m	m	m	m	m	m	m
<i>logname</i>	m	m	m	m	m	m	m
<i>lp</i>	m	m	m	m	m	m	m
<i>ls</i>	m	m	m	m	m	m	m
<i>m4</i>	m	m	d	xsi	d	d	.
<i>mailx</i>	m	m	m	m	m	m	m
<i>make</i>	d	sd	d	sd	d	d	o
<i>man</i>	m	m	m	m	m	m	o
<i>mesg</i>	m	m	m	up	m	m	o
<i>mkdir</i>	m	m	m	m	m	m	m
<i>mkfifo</i>	m	m	m	m	m	m	m
<i>more</i>	m	up	m	up	m	m	o
<i>mv</i>	m	m	m	m	m	m	m
<i>newgrp</i>	m	m	m	up	m	m	o
<i>nice</i>	m	m	m	up	m	m	o
<i>nl</i>	m	xsi	m	xsi	m	m	.
<i>nm</i>	d	xsi sd	d	sd up	d	d	o
<i>nohup</i>	m	m	m	m	m	m	m
<i>od</i>	m	m	m	m	m	m	m
<i>paste</i>	m	m	m	m	m	m	m
<i>patch</i>	m	m	m	up	m	m	o
<i>pathchk</i>	m	m	m	m	m	m	m
<i>pax</i>	m	m	m	m	m	m	m
<i>pr</i>	m	m	m	m	m	m	m
<i>printf</i>	m	m	m	m	m	m	m
<i>prs</i>	d	xsi	d	xsi	d	d	.
<i>ps</i>	m	m	m	up	m	m	o
<i>pwd</i>	m	m	m	m	m	m	m
<i>qalter</i>	ob o	ob be	o	be	.	.	2d
<i>qdel</i>	ob o	ob be	o	be	.	.	2d
<i>qhold</i>	ob o	ob be	o	be	.	.	2d
<i>qmove</i>	ob o	ob be	o	be	.	.	2d
<i>qmsg</i>	ob o	ob be	o	be	.	.	2d

Interface	XSI	POSIX	UNIX 03	POSIX 01	UNIX 98	UNIX 95	POSIX.2-92
<i>qrerun</i>	ob o	ob be	o	be	.	.	2d
<i>qrls</i>	ob o	ob be	o	be	.	.	2d
<i>qselect</i>	ob o	ob be	o	be	.	.	2d
<i>qsig</i>	ob o	ob be	o	be	.	.	2d
<i>qstat</i>	ob o	ob be	o	be	.	.	2d
<i>qsub</i>	ob o	ob be	o	be	.	.	2d
<i>read</i>	m	m	m	m	m	m	m
<i>renice</i>	m	m	m	up	m	m	o
<i>rm</i>	m	m	m	m	m	m	m
<i>rmdel</i>	d	xsi	d	xsi	d	d	.
<i>rmdir</i>	m	m	m	m	m	m	m
<i>sact</i>	d	xsi	d	xsi	d	d	.
<i>sccs</i>	d	xsi	d	xsi	d	d	.
<i>sed</i>	m	m	m	m	m	m	m
<i>sh</i>	m	m	m	m	m	m	m
<i>sleep</i>	m	m	m	m	m	m	m
<i>sort</i>	m	m	m	m	m	m	m
<i>split</i>	m	m	m	up	m	m	o
<i>strings</i>	m	m	m	up	m	m	o
<i>strip</i>	d	sd	d	sd	d	d	o
<i>stty</i>	m	m	m	m	m	m	m
<i>tabs</i>	m	m	m	up	m	m	o
<i>tail</i>	m	m	m	m	m	m	m
<i>talk</i>	m	up	m	up	m	m	o
<i>tee</i>	m	m	m	m	m	m	.
<i>test</i>	m	m	m	m	m	m	m
<i>time</i>	m	m	m	up	m	m	o
<i>touch</i>	m	m	m	m	m	m	m
<i>tput</i>	m	m	m	up	m	m	o
<i>tr</i>	m	m	m	m	m	m	m
<i>true</i>	m	m	m	m	m	m	m
<i>tsort</i>	m	m	m	xsi	m	m	.
<i>tty</i>	m	m	m	m	m	m	m
<i>type</i>	m	xsi	m	xsi	m	m	.
<i>ulimit</i>	m	xsi	m	xsi	m	m	.
<i>umask</i>	m	m	m	m	m	m	m
<i>unalias</i>	m	m	m	up	m	m	o
<i>uname</i>	m	m	m	m	m	m	m
<i>uncompress</i>	m	xsi	m	xsi	m	m	.
<i>unexpand</i>	m	m	m	up	m	m	o
<i>unget</i>	d	xsi	d	xsi	d	d	.
<i>uniq</i>	m	m	m	m	m	m	m
<i>unlink</i>	m	xsi	m	xsi	m	.	.
<i>uucp</i>	o	uu	m	xsi	m	m	.
<i>uudecode</i>	m	m	m	up	m	m	o
<i>uuencode</i>	m	m	m	up	m	m	o
<i>uustat</i>	o	uu	m	xsi	m	m	.
<i>uux</i>	o	uu	m	xsi	m	m	.
<i>val</i>	d	xsi	d	xsi	d	d	.
<i>vi</i>	m	up	m	up	m	m	o
<i>wait</i>	m	m	m	m	m	m	m

Interface	XSI	POSIX	UNIX 03	POSIX 01	UNIX 98	UNIX 95	POSIX.2-92
<i>wc</i>	m	m	m	m	m	m	m
<i>what</i>	d	xsi	d	xsi	d	d	.
<i>who</i>	m	m	m	up	m	m	o
<i>write</i>	m	m	m	up	m	m	o
<i>xargs</i>	m	m	m	m	m	m	m
<i>yacc</i>	d	cd	d	cd	d	d	o
<i>zcat</i>	m	xsi	m	xsi	m	m	.

Header Interface Table

9.1 Introduction

This chapter lists all the headers defined in XBD, Issue 7, complete with an indication of their status for XSI-conforming and POSIX-conforming systems, and their availability in UNIX 03, UNIX 98, UNIX 95, POSIX.1-2001 (denoted P01), the ISO POSIX-1: 1996 standard (denoted P96), the POSIX.2-1992 standard (denoted P92), the ISO C standard (denoted C99), and C89.

The following conventions are used in columns 2 through 10:

- m** Indicates that the interface is defined as mandatory.
- o** Indicates that the interface is part of an Option or Feature Group.
- ob** Indicates that the interface is Obsolescent, and although mandatory for the implementation, applications are discouraged from its use.
- ob o** In the XSI column, indicates that the interface is Obsolescent, and is part of an Option or Feature Group. Applications are discouraged from its use.
- opt* In the POSIX and P01 columns, two or three letter option codes are used as described in XBD, Issue 7 (for the POSIX column) and XBD, Issue 6 (for the P01 column), denoting the option to which the interface belongs.
- ob opt** Same as *opt*, but the interface is also obsolescent.
- r** In the P96 column, indicates that the interface is part of the POSIX Realtime Extension.
- t** In the P96 column, indicates that the interface is part of the POSIX Threads Extension.
- 1** In the C89 column, indicates that the interface is part of the ISO/IEC 9899: 1990/Amendment 1: 1995 (E).
- .** Indicates that the interface is not specified.

9.2 Header Interface Table

There are 82 headers listed.

Interface	XSI	POSIX	UNIX 03	P01	UNIX 98	UNIX 95	P96 P92	C99	C89
<aioh>	m	m	o	aio	o	.	r	.	.
<arpa/inet.h>	m	m	m	m	m	m	.	.	.
<assert.h>	m	m	m	m	m	m	.	m	m
<complex.h>	m	m	m	m	.	.	.	m	.
<cpio.h>	m	m	m	xsi	m	m	.	.	.
<ctype.h>	m	m	m	m	m	m	.	m	m
<dirent.h>	m	m	m	m	m	m	m	.	.
<dlfcn.h>	m	m	m	xsi	m
<errno.h>	m	m	m	m	m	m	m	m	m
<fcntl.h>	m	m	m	m	m	m	m	.	.
<fenv.h>	m	m	m	m	.	.	.	m	.
<float.h>	m	m	m	m	m	m	m	m	m
<fmtmsg.h>	m	xsi	m	xsi	m	m	m	.	.
<fnmatch.h>	m	m	m	m	m	m	m	.	.
<ftw.h>	m	xsi	m	xsi	m	m	.	.	.
<glob.h>	m	m	m	m	m	m	m	.	.
<grp.h>	m	m	m	m	m	m	m	.	.
<iconv.h>	m	m	m	xsi	m	m	.	.	.
<inttypes.h>	m	m	m	m	m	.	.	m	.
<iso646.h>	m	m	m	m	m	.	.	m	1
<langinfo.h>	m	m	m	xsi	m	m	.	.	.
<libgen.h>	m	xsi	m	xsi	m	m	.	.	.
<limits.h>	m	m	m	m	m	m	m	m	m
<locale.h>	m	m	m	m	m	m	m	m	m
<math.h>	m	m	m	m	m	m	m	m	m
<monetary.h>	m	m	m	xsi	m	m	.	.	.
<mqueue.h>	o	msg	o	msg	o	.	r	.	.
<ndbm.h>	m	xsi	m	xsi	m	m	.	.	.
<net/if.h>	m	m	m	m	m	m	.	.	.
<netdb.h>	m	m	m	m	m	m	m	.	.
<netinet/in.h>	m	m	m	m	m	m	m	.	.
<netinet/tcp.h>	m	m	m	m	m	m	m	.	.
<nI_types.h>	m	m	m	xsi	m	m	.	.	.
<poll.h>	m	m	m	m	m	m	.	.	.
<pthread.h>	m	m	m	thr	m	.	t	.	.
<pwd.h>	m	m	m	m	m	m	m	.	.
<regex.h>	m	m	m	m	m	m	m	.	.
<sched.h>	m	m	o	ps	o	.	r	.	.
<search.h>	m	xsi	m	xsi	m	m	.	.	.
<semaphore.h>	m	m	o	sem	o	.	r	.	.
<setjmp.h>	m	m	m	m	m	m	m	m	m
<signal.h>	m	m	m	m	m	m	m	m	m
<spawn.h>	o	spn	o	spn
<stdarg.h>	m	m	m	m	m	m	m	m	m
<stdbool.h>	m	m	m	m	m	m	m	m	.
<stddef.h>	m	m	m	m	m	m	m	m	m
<stdint.h>	m	m	m	m	.	.	.	m	.
<stdio.h>	m	m	m	m	m	m	m	m	m
<stdlib.h>	m	m	m	m	m	m	m	m	m
<string.h>	m	m	m	m	m	m	m	m	m
<strings.h>	m	m	m	xsi	m	m	.	.	.

Interface	XSI	POSIX	UNIX 03	P01	UNIX 98	UNIX 95	P96 P92	C99	C89
<stropts.h>	ob o	ob xsr	o	xsr	m	m	.	.	.
<sys/ipc.h>	m	xsi	m	xsi	m	m	.	.	.
<sys/mman.h>	m	m	m	m	m	m	m	.	.
<sys/msg.h>	m	xsi	m	xsi	m	m	.	.	.
<sys/resource.h>	m	xsi	m	xsi	m	m	.	.	.
<sys/select.h>	m	m	m	m	m	m	.	.	.
<sys/sem.h>	m	xsi	m	xsi	m	m	m	.	.
<sys/shm.h>	m	xsi	m	xsi	m	m	.	.	.
<sys/socket.h>	m	m	m	m	m	m	.	.	.
<sys/stat.h>	m	m	m	m	m	m	m	.	.
<sys/statvfs.h>	m	m	m	xsi	m	m	m	.	.
<sys/time.h>	m	xsi	m	xsi	m	m	.	.	.
<sys/times.h>	m	m	m	m	m	m	m	.	.
<sys/types.h>	m	m	m	m	m	m	m	.	.
<sys/uio.h>	m	xsi	m	xsi	m	m	.	.	.
<sys/un.h>	m	m	m	m	m	m	.	.	.
<sys/utsname.h>	m	m	m	m	m	m	m	.	.
<sys/wait.h>	m	m	m	m	m	m	m	.	.
<syslog.h>	m	xsi	m	xsi	m	m	.	.	.
<tar.h>	m	m	m	m	m	m	m	.	.
<termios.h>	m	m	m	m	m	m	m	.	.
<tgmath.h>	m	m	m	m	.	.	.	m	.
<time.h>	m	m	m	m	m	m	.	m	m
<trace.h>	ob o	ob trc	o	trc
<ulimit.h>	ob	ob xsi	m	xsi	m	m	.	.	.
<unistd.h>	m	m	m	m	m	m	m	.	.
<utime.h>	ob	ob	m	m	m	m	m	.	.
<utmpx.h>	m	xsi	m	xsi	m	m	.	.	.
<wchar.h>	m	m	m	m	m	m	.	m	1
<wctype.h>	m	m	m	m	m	m	.	m	1
<wordexp.h>	m	m	m	m	m	m	m	.	.

XCURSES Interface Table

10.1 Introduction

This chapter contains tables of all the interfaces defined in X/Open Curses, Issue 7, complete with an indication of their availability for the Single UNIX Specification (denoted by SUSv4), and their availability in UNIX 03, UNIX 98, UNIX 95, and Curses, Issue 3. The XCURSES interfaces are not specified for the POSIX base in POSIX.1-2008.

There is one table for system interfaces, one for headers, and one for utilities.

The following conventions are used in columns 2 through 6:

- m** Indicates that the interface is defined as mandatory.
- ob** Indicates that the interface is Obsolescent, and although mandatory for the implementation, applications are discouraged from its use.
- .** Indicates that the interface is not specified.

The tables are intended as a quick reference guide for programmers migrating to or developing applications for the Single UNIX Specification, Version 4.

10.2 XCURSES Interface Table

System Interfaces

There are 379 system interfaces listed.

Interface	SUSv4	UNIX 03	UNIX 98	UNIX 95	Curses 3
<i>COLOR_PAIR()</i>	m	m	m	m	.
<i>COLOR_PAIRS</i>	m	m	m	m	.
<i>COLORS</i>	m	m	m	m	.
<i>COLS</i>	m	m	m	m	m
<i>LINES</i>	m	m	m	m	m
<i>PAIR_NUMBER()</i>	m	m	m	m	.
<i>add_wch()</i>	m	m	m	m	.
<i>add_wchnstr()</i>	m	m	m	m	.
<i>add_wchstr()</i>	m	m	m	m	.
<i>addch()</i>	m	m	m	m	m
<i>addchnstr()</i>	m	m	m	m	.

Interface	SUSv4	UNIX 03	UNIX 98	UNIX 95	Curses 3
<i>addchstr()</i>	m	m	m	m	.
<i>addnstr()</i>	m	m	m	m	.
<i>addnwstr()</i>	m	m	m	m	.
<i>addstr()</i>	m	m	m	m	m
<i>addwstr()</i>	m	m	m	m	.
<i>attr_get()</i>	m	m	m	m	.
<i>attr_off()</i>	m	m	m	m	.
<i>attr_on()</i>	m	m	m	m	.
<i>attr_set()</i>	m	m	m	m	.
<i>attroff()</i>	m	m	m	m	m
<i>attron()</i>	m	m	m	m	m
<i>attrset()</i>	m	m	m	m	m
<i>baudrate()</i>	m	m	m	m	m
<i>beep()</i>	m	m	m	m	m
<i>bkgd()</i>	m	m	m	m	.
<i>bkgdset()</i>	m	m	m	m	.
<i>bkgrnd()</i>	m	m	m	m	.
<i>bkgrndset()</i>	m	m	m	m	.
<i>border()</i>	m	m	m	m	.
<i>border_set()</i>	m	m	m	m	.
<i>box()</i>	m	m	m	m	m
<i>box_set()</i>	m	m	m	m	.
<i>can_change_color()</i>	m	m	m	m	.
<i>cbreak()</i>	m	m	m	m	m
<i>chgat()</i>	m	m	m	m	.
<i>clear()</i>	m	m	m	m	m
<i>clearok()</i>	m	m	m	m	m
<i>clrtoebot()</i>	m	m	m	m	m
<i>clrtoeol()</i>	m	m	m	m	m
<i>color_content()</i>	m	m	m	m	.
<i>color_set()</i>	m	m	m	m	.
<i>copywin()</i>	m	m	m	m	.
<i>cur_term</i>	m	m	m	m	.
<i>curs_set()</i>	m	m	m	m	.
<i>curscr</i>	m	m	m	m	.
<i>def_prog_mode()</i>	m	m	m	m	m
<i>def_shell_mode()</i>	m	m	m	m	m
<i>del_curterm()</i>	m	m	m	m	.
<i>delay_output()</i>	m	m	m	m	m
<i>delch()</i>	m	m	m	m	m
<i>deleteln()</i>	m	m	m	m	m
<i>delscreen()</i>	m	m	m	m	.
<i>delwin()</i>	m	m	m	m	m
<i>derwin()</i>	m	m	m	m	.
<i>doupdate()</i>	m	m	m	m	m
<i>dupwin()</i>	m	m	m	m	.
<i>echo()</i>	m	m	m	m	m
<i>echo_wchar()</i>	m	m	m	m	.
<i>echochar()</i>	m	m	m	m	.
<i>endwin()</i>	m	m	m	m	m
<i>erase()</i>	m	m	m	m	m

Interface	SUSv4	UNIX 03	UNIX 98	UNIX 95	Curses 3
<i>erasechar()</i>	m	m	m	m	m
<i>erasewchar()</i>	m	m	m	m	.
<i>filter()</i>	m	m	m	m	.
<i>flash()</i>	m	m	m	m	m
<i>flushinp()</i>	m	m	m	m	m
<i>get_wch()</i>	m	m	m	m	.
<i>get_wstr()</i>	m	m	m	m	.
<i>getbegyx()</i>	m	m	m	m	.
<i>getbkgd()</i>	m	m	m	m	.
<i>getbkgrnd()</i>	m	m	m	m	.
<i>getcchar()</i>	m	m	m	m	.
<i>getch()</i>	m	m	m	m	m
<i>getmaxyx()</i>	m	m	m	m	.
<i>getn_wstr()</i>	m	m	m	m	.
<i>getnstr()</i>	m	m	m	m	.
<i>getparyx()</i>	m	m	m	m	.
<i>getstr()</i>	m	m	m	m	m
<i>getwin()</i>	m	m	m	m	.
<i>getyx()</i>	m	m	m	m	m
<i>halfdelay()</i>	m	m	m	m	.
<i>has_colors()</i>	m	m	m	m	.
<i>has_ic()</i>	m	m	m	m	m
<i>has_il()</i>	m	m	m	m	m
<i>hline()</i>	m	m	m	m	.
<i>hline_set()</i>	m	m	m	m	.
<i>idcok()</i>	m	m	m	m	.
<i>idlok()</i>	m	m	m	m	m
<i>immedok()</i>	m	m	m	m	.
<i>in_wch()</i>	m	m	m	m	.
<i>in_wchnstr()</i>	m	m	m	m	.
<i>in_wchstr()</i>	m	m	m	m	.
<i>inch()</i>	m	m	m	m	m
<i>inchnstr()</i>	m	m	m	m	.
<i>inchstr()</i>	m	m	m	m	.
<i>init_color()</i>	m	m	m	m	.
<i>init_pair()</i>	m	m	m	m	.
<i>initscr()</i>	m	m	m	m	m
<i>innstr()</i>	m	m	m	m	.
<i>innwstr()</i>	m	m	m	m	.
<i>ins_nwstr()</i>	m	m	m	m	.
<i>ins_wch()</i>	m	m	m	m	.
<i>ins_wstr()</i>	m	m	m	m	.
<i>insch()</i>	m	m	m	m	m
<i>insdelIn()</i>	m	m	m	m	.
<i>insertIn()</i>	m	m	m	m	m
<i>insnstr()</i>	m	m	m	m	.
<i>insstr()</i>	m	m	m	m	.
<i>instr()</i>	m	m	m	m	.
<i>intrflush()</i>	m	m	m	m	m
<i>inwstr()</i>	m	m	m	m	.
<i>is_linetouched()</i>	m	m	m	m	.

Interface	SUSv4	UNIX 03	UNIX 98	UNIX 95	Curses 3
<i>is_wintouched()</i>	m	m	m	m	.
<i>isendwin()</i>	m	m	m	m	.
<i>key_name()</i>	m	m	m	m	.
<i>keyname()</i>	m	m	m	m	.
<i>keypad()</i>	m	m	m	m	m
<i>killchar()</i>	m	m	m	m	m
<i>killwchar()</i>	m	m	m	m	.
<i>leaveok()</i>	m	m	m	m	m
<i>longname()</i>	m	m	m	m	m
<i>meta()</i>	m	m	m	m	.
<i>move()</i>	m	m	m	m	m
<i>mvadd_wch()</i>	m	m	m	m	.
<i>mvadd_wchnstr()</i>	m	m	m	m	.
<i>mvadd_wchstr()</i>	m	m	m	m	.
<i>mvaddch()</i>	m	m	m	m	m
<i>mvaddchnstr()</i>	m	m	m	m	.
<i>mvaddchstr()</i>	m	m	m	m	.
<i>mvaddnstr()</i>	m	m	m	m	.
<i>mvaddnwstr()</i>	m	m	m	m	.
<i>mvaddstr()</i>	m	m	m	m	m
<i>mvaddwstr()</i>	m	m	m	m	.
<i>mvchgat()</i>	m	m	m	m	.
<i>mvcur()</i>	m	m	m	m	.
<i>mvdelch()</i>	m	m	m	m	m
<i>mvderwin()</i>	m	m	m	m	.
<i>mvget_wch()</i>	m	m	m	m	.
<i>mvget_wstr()</i>	m	m	m	m	.
<i>mvgetch()</i>	m	m	m	m	m
<i>mvgetn_wstr()</i>	m	m	m	m	.
<i>mvgetnstr()</i>	m	m	m	m	.
<i>mvgetstr()</i>	m	m	m	m	m
<i>mvhline()</i>	m	m	m	m	.
<i>mvhline_set()</i>	m	m	m	m	.
<i>mvin_wch()</i>	m	m	m	m	.
<i>mvin_wchnstr()</i>	m	m	m	m	.
<i>mvin_wchstr()</i>	m	m	m	m	.
<i>mvinch()</i>	m	m	m	m	m
<i>mvinchnstr()</i>	m	m	m	m	.
<i>mvinchstr()</i>	m	m	m	m	.
<i>mvinnstr()</i>	m	m	m	m	.
<i>mvinnwstr()</i>	m	m	m	m	.
<i>mvins_nwstr()</i>	m	m	m	m	.
<i>mvins_wch()</i>	m	m	m	m	.
<i>mvins_wstr()</i>	m	m	m	m	.
<i>mvinsch()</i>	m	m	m	m	m
<i>mvinsnstr()</i>	m	m	m	m	.
<i>mvinsstr()</i>	m	m	m	m	.
<i>mvinstr()</i>	m	m	m	m	.
<i>mvinswstr()</i>	m	m	m	m	.
<i>mvprintw()</i>	m	m	m	m	m
<i>mvscanw()</i>	m	m	m	m	m

Interface	SUSv4	UNIX 03	UNIX 98	UNIX 95	Curses 3
<i>mvvline()</i>	m	m	m	m	.
<i>mvvline_set()</i>	m	m	m	m	.
<i>mvwadd_wch()</i>	m	m	m	m	.
<i>mvwadd_wchnstr()</i>	m	m	m	m	.
<i>mvwadd_wchstr()</i>	m	m	m	m	.
<i>mvwaddch()</i>	m	m	m	m	m
<i>mvwaddchnstr()</i>	m	m	m	m	.
<i>mvwaddchstr()</i>	m	m	m	m	.
<i>mvwaddnstr()</i>	m	m	m	m	.
<i>mvwaddnwstr()</i>	m	m	m	m	.
<i>mvwaddstr()</i>	m	m	m	m	m
<i>mvwaddwstr()</i>	m	m	m	m	.
<i>mvwchgat()</i>	m	m	m	m	.
<i>mvwdelch()</i>	m	m	m	m	m
<i>mvwget_wch()</i>	m	m	m	m	.
<i>mvwget_wstr()</i>	m	m	m	m	.
<i>mvwgetch()</i>	m	m	m	m	m
<i>mvwgetn_wstr()</i>	m	m	m	m	.
<i>mvwgetnstr()</i>	m	m	m	m	.
<i>mvwgetstr()</i>	m	m	m	m	m
<i>mvwhline()</i>	m	m	m	m	.
<i>mvwhline_set()</i>	m	m	m	m	.
<i>mvwin()</i>	m	m	m	m	m
<i>mvwin_wch()</i>	m	m	m	m	.
<i>mvwin_wchnstr()</i>	m	m	m	m	.
<i>mvwin_wchstr()</i>	m	m	m	m	.
<i>mvwinch()</i>	m	m	m	m	m
<i>mvwinchnstr()</i>	m	m	m	m	.
<i>mvwinchstr()</i>	m	m	m	m	.
<i>mvwinnstr()</i>	m	m	m	m	.
<i>mvwinnwstr()</i>	m	m	m	m	.
<i>mvwins_nwstr()</i>	m	m	m	m	.
<i>mvwins_wch()</i>	m	m	m	m	.
<i>mvwins_wstr()</i>	m	m	m	m	.
<i>mvwinsch()</i>	m	m	m	m	m
<i>mvwinsnstr()</i>	m	m	m	m	.
<i>mvwinsstr()</i>	m	m	m	m	.
<i>mvwinstr()</i>	m	m	m	m	.
<i>mvwinwstr()</i>	m	m	m	m	.
<i>mvwprintw()</i>	m	m	m	m	m
<i>mvwscanw()</i>	m	m	m	m	m
<i>mvwvline()</i>	m	m	m	m	.
<i>mvwvline_set()</i>	m	m	m	m	.
<i>napms()</i>	m	m	m	m	.
<i>newpad()</i>	m	m	m	m	m
<i>newterm()</i>	m	m	m	m	m
<i>newwin()</i>	m	m	m	m	m
<i>nl()</i>	m	m	m	m	m
<i>nocbreak()</i>	m	m	m	m	m
<i>nodelay()</i>	m	m	m	m	m
<i>noecho()</i>	m	m	m	m	m

Interface	SUSv4	UNIX 03	UNIX 98	UNIX 95	Curses 3
<i>nonl()</i>	m	m	m	m	m
<i>noqiflush()</i>	m	m	m	m	.
<i>noraw()</i>	m	m	m	m	m
<i>notimeout()</i>	m	m	m	m	.
<i>overlay()</i>	m	m	m	m	m
<i>overwrite()</i>	m	m	m	m	m
<i>pair_content()</i>	m	m	m	m	.
<i>pecho_wchar()</i>	m	m	m	m	.
<i>pechochar()</i>	m	m	m	m	.
<i>pnoutrefresh()</i>	m	m	m	m	m
<i>prefresh()</i>	m	m	m	m	m
<i>printw()</i>	m	m	m	m	m
<i>putp()</i>	m	m	m	m	.
<i>putwin()</i>	m	m	m	m	.
<i>qiflush()</i>	m	m	m	m	.
<i>raw()</i>	m	m	m	m	m
<i>redrawwin()</i>	m	m	m	m	.
<i>refresh()</i>	m	m	m	m	m
<i>reset_prog_mode()</i>	m	m	m	m	m
<i>reset_shell_mode()</i>	m	m	m	m	m
<i>resetty()</i>	m	m	m	m	m
<i>restartterm()</i>	m	m	m	m	.
<i>ripoffline()</i>	m	m	m	m	.
<i>savetty()</i>	m	m	m	m	m
<i>scanw()</i>	m	m	m	m	m
<i>scr_dump()</i>	m	m	m	m	.
<i>scr_init()</i>	m	m	m	m	.
<i>scr_restore()</i>	m	m	m	m	.
<i>scr_set()</i>	m	m	m	m	.
<i>scr1()</i>	m	m	m	m	.
<i>scroll()</i>	m	m	m	m	m
<i>scrollok()</i>	m	m	m	m	m
<i>set_curterm()</i>	m	m	m	m	.
<i>set_term()</i>	m	m	m	m	m
<i>setcchar()</i>	m	m	m	m	.
<i>setscreg()</i>	m	m	m	m	m
<i>setupterm()</i>	m	m	m	m	.
<i>slk_attr_off()</i>	m	m	m	m	.
<i>slk_attr_on()</i>	m	m	m	m	.
<i>slk_attr_set()</i>	m	m	m	m	.
<i>slk_attroff()</i>	m	m	m	m	.
<i>slk_attron()</i>	m	m	m	m	.
<i>slk_attrset()</i>	m	m	m	m	.
<i>slk_clear()</i>	m	m	m	m	.
<i>slk_color()</i>	m	m	m	m	.
<i>slk_init()</i>	m	m	m	m	.
<i>slk_label()</i>	m	m	m	m	.
<i>slk_noutrefresh()</i>	m	m	m	m	.
<i>slk_refresh()</i>	m	m	m	m	.
<i>slk_restore()</i>	m	m	m	m	.
<i>slk_set()</i>	m	m	m	m	.

Interface	SUSv4	UNIX 03	UNIX 98	UNIX 95	Curses 3
<i>slk_touch()</i>	m	m	m	m	.
<i>slk_wset()</i>	m	m	m	m	.
<i>standend()</i>	m	m	m	m	m
<i>standout()</i>	m	m	m	m	m
<i>start_color()</i>	m	m	m	m	.
<i>stdscr</i>	m	m	m	m	m
<i>subpad()</i>	m	m	m	m	.
<i>subwin()</i>	m	m	m	m	m
<i>syncok()</i>	m	m	m	m	.
<i>term_attrs()</i>	m	m	m	m	.
<i>termattrs()</i>	m	m	m	m	.
<i>termname()</i>	m	m	m	m	.
<i>tigetflag()</i>	m	m	m	m	.
<i>tigetnum()</i>	m	m	m	m	.
<i>tigetstr()</i>	m	m	m	m	.
<i>timeout()</i>	m	m	m	m	.
<i>tiparm()</i>	m
<i>touchline()</i>	m	m	m	m	.
<i>touchwin()</i>	m	m	m	m	m
<i>tparm()</i>	ob	m	m	m	.
<i>tputs()</i>	m	m	m	m	.
<i>typeahead()</i>	m	m	m	m	m
<i>unctrl()</i>	m	m	m	m	m
<i>unget_wch()</i>	m	m	m	m	.
<i>ungetch()</i>	m	m	m	m	.
<i>untouchwin()</i>	m	m	m	m	.
<i>use_env()</i>	m	m	m	m	.
<i>vid_attr()</i>	m	m	m	m	.
<i>vid_puts()</i>	m	m	m	m	.
<i>vidattr()</i>	m	m	m	m	.
<i>vidputs()</i>	m	m	m	m	.
<i>vline()</i>	m	m	m	m	.
<i>vline_set()</i>	m	m	m	m	.
<i>vw_printw()</i>	m	m	m	m	.
<i>vw_scanw()</i>	m	m	m	m	.
<i>wadd_wch()</i>	m	m	m	m	.
<i>wadd_wchnstr()</i>	m	m	m	m	.
<i>wadd_wchstr()</i>	m	m	m	m	.
<i>waddch()</i>	m	m	m	m	m
<i>waddchnstr()</i>	m	m	m	m	.
<i>waddchstr()</i>	m	m	m	m	.
<i>waddnstr()</i>	m	m	m	m	.
<i>waddnwstr()</i>	m	m	m	m	.
<i>waddstr()</i>	m	m	m	m	m
<i>waddwstr()</i>	m	m	m	m	.
<i>wattr_get()</i>	m	m	m	m	.
<i>wattr_off()</i>	m	m	m	m	.
<i>wattr_on()</i>	m	m	m	m	.
<i>wattr_set()</i>	m	m	m	m	.
<i>wattroff()</i>	m	m	m	m	m
<i>wattron()</i>	m	m	m	m	m

Interface	SUSv4	UNIX 03	UNIX 98	UNIX 95	Curses 3
<i>wattrset()</i>	m	m	m	m	m
<i>wbkgd()</i>	m	m	m	m	.
<i>wbkgdset()</i>	m	m	m	m	.
<i>wbkgnd()</i>	m	m	m	m	.
<i>wbkgndset()</i>	m	m	m	m	.
<i>wborder()</i>	m	m	m	m	.
<i>wborder_set()</i>	m	m	m	m	.
<i>wchgat()</i>	m	m	m	m	.
<i>wclear()</i>	m	m	m	m	m
<i>wclrtoobot()</i>	m	m	m	m	m
<i>wclrtoeol()</i>	m	m	m	m	m
<i>wcolor_set()</i>	m	m	m	m	.
<i>wcursyncup()</i>	m	m	m	m	.
<i>wdelch()</i>	m	m	m	m	m
<i>wdeleteln()</i>	m	m	m	m	m
<i>wecho_wchar()</i>	m	m	m	m	.
<i>wechochar()</i>	m	m	m	m	.
<i>werase()</i>	m	m	m	m	m
<i>wget_wch()</i>	m	m	m	m	.
<i>wget_wstr()</i>	m	m	m	m	.
<i>wgetbkgrnd()</i>	m	m	m	m	.
<i>wgetch()</i>	m	m	m	m	m
<i>wgetn_wstr()</i>	m	m	m	m	.
<i>wgetnstr()</i>	m	m	m	m	.
<i>wgetstr()</i>	m	m	m	m	m
<i>whline()</i>	m	m	m	m	.
<i>whline_set()</i>	m	m	m	m	.
<i>win_wch()</i>	m	m	m	m	.
<i>win_wchnstr()</i>	m	m	m	m	.
<i>win_wchstr()</i>	m	m	m	m	.
<i>winch()</i>	m	m	m	m	m
<i>winchnstr()</i>	m	m	m	m	.
<i>winchstr()</i>	m	m	m	m	.
<i>winnstr()</i>	m	m	m	m	.
<i>winnwstr()</i>	m	m	m	m	.
<i>wins_nwstr()</i>	m	m	m	m	.
<i>wins_wch()</i>	m	m	m	m	.
<i>wins_wstr()</i>	m	m	m	m	.
<i>winsch()</i>	m	m	m	m	m
<i>winsdelln()</i>	m	m	m	m	.
<i>winsertln()</i>	m	m	m	m	m
<i>winsnstr()</i>	m	m	m	m	.
<i>winsstr()</i>	m	m	m	m	.
<i>winstr()</i>	m	m	m	m	.
<i>winwstr()</i>	m	m	m	m	.
<i>wmove()</i>	m	m	m	m	m
<i>wnoutrefresh()</i>	m	m	m	m	m
<i>wprintw()</i>	m	m	m	m	m
<i>wredrawln()</i>	m	m	m	m	.
<i>wrefresh()</i>	m	m	m	m	m
<i>wscanw()</i>	m	m	m	m	m

Interface	SUSv4	UNIX 03	UNIX 98	UNIX 95	Curses 3
<i>wscr()</i>	m	m	m	m	.
<i>wsetscrreg()</i>	m	m	m	m	m
<i>wstandend()</i>	m	m	m	m	m
<i>wstandout()</i>	m	m	m	m	m
<i>wsyncdown()</i>	m	m	m	m	.
<i>wsyncup()</i>	m	m	m	m	.
<i>wtimeout()</i>	m	m	m	m	.
<i>wtouchln()</i>	m	m	m	m	.
<i>wunctrl()</i>	m	m	m	m	.
<i>wvline()</i>	m	m	m	m	.
<i>wvline_set()</i>	m	m	m	m	.

Headers

Interface	SUSv4	UNIX 03	UNIX 98	UNIX 95	Curses 3
< curses.h >	m	m	m	m	m
< term.h >	m	m	m	m	.
< unctrl.h >	m	m	m	m	m

Utilities

Note that the *tput* entry here is for the fully-specified *tput* utility in X/Open Curses, Issue 7, not the minimally-specified *tput* utility in XCU, Issue 7, which has an entry in the table in [Chapter 8](#) (on page 81).

Interface	SUSv4	UNIX 03	UNIX 98	UNIX 95	Curses 3
<i>infocmp</i>	m
<i>tic</i>	m
<i>tput</i>	m
<i>untic</i>	m

System Interfaces Migration

11.1 Introduction

This chapter contains a section for each system interface defined in XSH, Issue 7. Each section contains the SYNOPSIS and gives the derivation of the interface. For interfaces new to Issue 7, a brief description is included, complete with examples where appropriate. For interfaces carried forward from Issue 6, syntax and semantic changes made to the interface in Issue 7 are identified (if any). Only changes that might affect an application programmer are included.

11.2 System Interfaces

_Exit, _exit

Purpose: Terminate a process.

Synopsis:

```
#include <stdlib.h>

void _Exit(int status);

#include <unistd.h>

void _exit(int status);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #085 is applied, clarifying the text regarding flushing of streams and closing of temporary files.

Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, and Semaphores options is moved to the Base.

_longjmp, _setjmp

Purpose: Non-local goto.

OB XSI Synopsis:

```
#include <setjmp.h>

void _longjmp(jmp_buf env, int val);
int _setjmp(jmp_buf env);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The `_longjmp()` and `_setjmp()` functions are marked obsolescent. Applications should use `siglongjmp()` and `sigsetjmp()` respectively.

`_tolower`

Purpose: Transliterate uppercase characters to lowercase.

OB XSI Synopsis:

```
#include <ctype.h>
int _tolower(int c);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The `_tolower()` function is marked obsolescent. Applications should use the `tolower()` function instead.

`_toupper`

Purpose: Transliterate lowercase characters to uppercase.

OB XSI Synopsis:

```
#include <ctype.h>
int _toupper(int c);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The `_toupper()` function is marked obsolescent. Applications should use the `toupper()` function instead.

`a64l, l64a`

Purpose: Convert between a 32-bit integer and a radix-64 ASCII string.

XSI Synopsis:

```
#include <stdlib.h>
long a64l(const char *s);
char *l64a(long value);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

`abort`

Purpose: Generate an abnormal process abort.

Synopsis:

```
#include <stdlib.h>
void abort(void);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

abs

Purpose: Return an integer absolute value.

Synopsis: `#include <stdlib.h>`
`int abs(int i);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

accept

Purpose: Accept a new connection on a socket.

Synopsis: `#include <sys/socket.h>`
`int accept(int socket, struct sockaddr *restrict address,`
`socklen_t *restrict address_len);`

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [ENOBUFS] and [ENOMEM] errors to become “shall fail” errors.

Functionality relating to XSI STREAMS is marked obsolescent.

access, faccessat

Purpose: Determine accessibility of a file relative to directory file descriptor.

Synopsis: `#include <unistd.h>`
`int access(const char *path, int amode);`
`int faccessat(int fd, const char *path, int amode, int flag);`

The `faccessat()` function is equivalent to the `access()` function, except in the case where `path` specifies a relative path. In this case the file whose accessibility is to be determined is located relative to the directory associated with the file descriptor `fd` instead of the current working directory. If the file descriptor was opened without `O_SEARCH`, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with `O_SEARCH`, the function does not perform the check.

The `AT_EACCESS` flag can be used to specify that checks for accessibility are performed using the effective user and group IDs instead of the real user and group ID.

The purpose of the `faccessat()` function is to enable the checking of the accessibility of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to `access()`, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the `faccessat()` function it can be guaranteed that the file tested for accessibility is located relative to the desired directory.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

- Issue 7: Austin Group Interpretation 1003.1-2001 #046 is applied.
- Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.
- The *faccessat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.
- The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

acos, acosf, acosl

Purpose: Arc cosine functions.

Synopsis: `#include <math.h>`

```
double acos(double x);
float acosf(float x);
long double acosl(long double x);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

acosh, acoshf, acoshl

Purpose: Inverse hyperbolic cosine functions.

Synopsis: `#include <math.h>`

```
double acosh(double x);
float acoshf(float x);
long double acoshl(long double x);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

aio_cancel

Purpose: Cancel an asynchronous I/O request.

Synopsis: `#include <aio.h>`

```
int aio_cancel(int fildes, struct aiocb *aiocbp);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: The *aio_cancel()* function is moved from the Asynchronous Input and Output option to the Base.

aio_error

Purpose: Retrieve errors status for an asynchronous I/O operation.

Synopsis:

```
#include <aio.h>

int aio_error(const struct aiocb *aiocbp);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: Austin Group Interpretation 1003.1-2001 #045 is applied, clarifying that the behavior is undefined if the **aiocb** structure pointed to by *aiocbp* is not associated with an operation that has been scheduled.

SD5-XSH-ERN-148 is applied, clarifying that when *aio_error()* fails it returns `-1` and sets *errno*.

The *aio_error()* function is moved from the Asynchronous Input and Output option to the Base.

aio_fsync

Purpose: Asynchronous file synchronization.

Synopsis:

```
#include <aio.h>

int aio_fsync(int op, struct aiocb *aiocbp);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: The *aio_fsync()* function is moved from the Asynchronous Input and Output option to the Base.

aio_read

Purpose: Asynchronous read from a file.

Synopsis:

```
#include <aio.h>

int aio_read(struct aiocb *aiocbp);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: Austin Group Interpretation 1003.1-2001 #082 is applied.

The *aio_read()* function is moved from the Asynchronous Input and Output option to the Base.

aio_return

Purpose: Retrieve return status of an asynchronous I/O operation.

Synopsis:

```
#include <aio.h>

ssize_t aio_return(struct aiocb *aiocbp);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: SD5-XSH-ERN-148 is applied, clarifying that when *aio_return()* fails it returns `-1` and sets *errno*.

The *aio_return()* function is moved from the Asynchronous Input and Output option to the Base.

aio_suspend

Purpose: Wait for an asynchronous I/O request.

Synopsis: `#include <aio.h>`

```
int aio_suspend(const struct aiocb *const list[], int nent,
               const struct timespec *timeout);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: The *aio_suspend()* function is moved from the Asynchronous Input and Output option to the Base.

aio_write

Purpose: Asynchronous write to a file.

Synopsis: `#include <aio.h>`

```
int aio_write(struct aiocb *aiocbp);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: Austin Group Interpretation 1003.1-2001 #082 is applied.

The *aio_write()* function is moved from the Asynchronous Input and Output option to the Base.

alarm

Purpose: Schedule an alarm signal.

Synopsis: `#include <unistd.h>`

```
unsigned alarm(unsigned seconds);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

alphasort, scandir

Purpose: Scan a directory.

Synopsis: `#include <dirent.h>`

```
int alphasort(const struct dirent **d1,
             const struct dirent **d2);
int scandir(const char *dir, struct dirent ***namelist,
            int (*sel)(const struct dirent *),
            int (*compar)(const struct dirent **,
                          const struct dirent **));
```

The *alphasort()* function can be used as the comparison function for the *scandir()*

function to sort the directory entries, *d1* and *d2*, into alphabetical order. Sorting happens as if by calling the *strcoll()* function on the *d_name* element of the **dirent** structures passed as the two parameters. If the *strcoll()* function fails, the return value of *alphasort()* is unspecified.

The *scandir()* function scans the directory *dir*, calling the function referenced by *sel* on each directory entry. Entries for which the function referenced by *sel* returns non-zero are stored in strings allocated as if by a call to *malloc()*, and sorted as if by a call to *qsort()* with the comparison function *compar*, except that *compar* need not provide total ordering. The strings are collected in array *namelist* which is allocated as if by a call to *malloc()*.

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Issue 7: First released in Issue 7.

asctime, asctime_r

Purpose: Convert date and time to a string.

OB Synopsis:

```
#include <time.h>

char *asctime(const struct tm *timeptr);
OB CX char *asctime_r(const struct tm *restrict tm,
char *restrict buf);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #053 is applied, marking these functions obsolescent. Applications should use the *strftime()* function instead.

The *asctime_r()* function is moved from the Thread-Safe Functions option to the Base.

asin, asinf, asinl

Purpose: Arc sine function.

Synopsis:

```
#include <math.h>

double asin(double x);
float asinf(float x);
long double asinl(long double x);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

asinh, asinhf, asinhl

Purpose: Inverse hyperbolic sine functions.

Synopsis:

```
#include <math.h>

double asinh(double x);
float asinhf(float x);
long double asinhl(long double x);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

assert

Purpose: Insert program diagnostics.

Synopsis:

```
#include <assert.h>

void assert(scalar expression);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

atan, atanf, atanl

Purpose: Arc tangent function.

Synopsis:

```
#include <math.h>

double atan(double x);
float atanf(float x);
long double atanl(long double x);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

atan2, atan2f, atan2l

Purpose: Arc tangent functions.

Synopsis:

```
#include <math.h>

double atan2(double y, double x);
float atan2f(float y, float x);
long double atan2l(long double y, long double x);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

atanh, atanhf, atanh1

Purpose: Inverse hyperbolic tangent functions.

Synopsis:

```
#include <math.h>

double atanh(double x);
float atanhf(float x);
long double atanh1(long double x);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

atexit

Purpose: Register a function to run at process termination.

Synopsis: `#include <stdlib.h>`
`int atexit(void (*func)(void));`

Derivation: First released in Issue 4. Derived from the ANSI C standard.

Issue 7: No functional changes are made in this issue.

atof

Purpose: Convert a string to a double-precision number.

Synopsis: `#include <stdlib.h>`
`double atof(const char *str);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

atoi

Purpose: Convert a string to an integer.

Synopsis: `#include <stdlib.h>`
`int atoi(const char *str);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

atol, atoll

Purpose: Convert a string to a long integer.

Synopsis: `#include <stdlib.h>`
`long atol(const char *str);`
`long long atoll(const char *nptr);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

basename

Purpose: Return the last component of a pathname.

XSI Synopsis: `#include <libgen.h>`
`char *basename(char *path);`

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

bind

Purpose: Bind a name to a socket.

Synopsis:

```
#include <sys/socket.h>

int bind(int socket, const struct sockaddr *address,
         socklen_t address_len);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [ENOBUFFS] error to become a “shall fail” error.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

SD5-XSH-ERN-185 is applied, specifying asynchronous behavior for *bind()* when *O_NONBLOCK* is set for the socket.

An example is added.

bsearch

Purpose: Binary search a sorted table.

Synopsis:

```
#include <stdlib.h>

void *bsearch(const void *key, const void *base, size_t nel,
             size_t width, int (*compar)(const void *, const void *));
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The EXAMPLES section is revised.

btowc

Purpose: Single byte to wide character conversion.

Synopsis:

```
#include <stdio.h>
#include <wchar.h>

wint_t btowc(int c);
```

Derivation: First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: No functional changes are made in this issue.

cabs, cabsf, cabsl

Purpose: Return a complex absolute value.

Synopsis:

```
#include <complex.h>

double cabs(double complex z);
float cabsf(float complex z);
long double cabsl(long double complex z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

Issue 7: No functional changes are made in this issue.

ccos, ccosf, ccosl

Purpose: Complex arc cosine functions.

Synopsis: `#include <complex.h>`

```
double complex ccos(double complex z);
float complex ccosf(float complex z);
long double complex ccosl(long double complex z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

cacosh, cacoshf, cacoshl

Purpose: Complex arc hyperbolic cosine functions.

Synopsis: `#include <complex.h>`

```
double complex cacosh(double complex z);
float complex cacoshf(float complex z);
long double complex cacoshl(long double complex z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

calloc

Purpose: A memory allocator.

Synopsis: `#include <stdlib.h>`

```
void *calloc(size_t nelem, size_t elsize);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

carg, cargf, cargl

Purpose: Complex argument functions.

Synopsis: `#include <complex.h>`

```
double carg(double complex z);
float cargf(float complex z);
long double cargl(long double complex z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

casin, casinf, casinl

Purpose: Complex arc sine functions.

Synopsis: `#include <complex.h>`

```
double complex casin(double complex z);
float complex casinf(float complex z);
long double complex casinl(long double complex z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

casinh, casinhf, casinhl

Purpose: Complex arc hyperbolic sine functions.

Synopsis: `#include <complex.h>`

```
double complex casinh(double complex z);
float complex casinhf(float complex z);
long double complex casinhl(long double complex z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

catan, catanf, catanl

Purpose: Complex arc tangent functions.

Synopsis: `#include <complex.h>`

```
double complex catan(double complex z);
float complex catanf(float complex z);
long double complex catanl(long double complex z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

catanh, catanhf, catanhl

Purpose: Complex arc hyperbolic tangent functions.

Synopsis: `#include <complex.h>`

```
double complex catanh(double complex z);
float complex catanhf(float complex z);
long double complex catanhl(long double complex z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

catclose

Purpose: Close a message catalog descriptor.

Synopsis: `#include <nl_types.h>`

```
int catclose(nl_catd catd);
```

Derivation: First released in Issue 2.

Issue 7: The `catclose()` function is moved from the XSI option to the Base.

catgets

Purpose: Read a program message.

Synopsis: `#include <nl_types.h>`
`char *catgets(nl_catd catd, int set_id, int msg_id,`
`const char *s);`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [EINTR] and [ENOMSG] errors to become “shall fail” errors, updating the RETURN VALUE section, and updating the DESCRIPTION to note that the results are undefined if *catd* is not a value returned by *catopen()* for a message catalog still open in the process.

The *catgets()* function is moved from the XSI option to the Base.

catopen

Purpose: Open a message catalog.

Synopsis: `#include <nl_types.h>`
`nl_catd catopen(const char *name, int oflag);`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

The *catopen()* function is moved from the XSI option to the Base.

cbrt, cbrtf, cbrtl

Purpose: Cube root functions.

Synopsis: `#include <math.h>`
`double cbrt(double x);`
`float cbrtf(float x);`
`long double cbrtl(long double x);`

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

ccos, ccosf, ccosl

Purpose: Complex cosine functions.

Synopsis: `#include <complex.h>`
`double complex ccos(double complex z);`
`float complex ccosf(float complex z);`
`long double complex ccosl(long double complex z);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

ccosh, ccoshf, ccoshl

Purpose: Complex hyperbolic cosine functions.

Synopsis:

```
#include <complex.h>

double complex ccosh(double complex z);
float complex ccoshf(float complex z);
long double complex ccoshl(long double complex z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

ceil, ceilf, ceill

Purpose: Ceiling value function.

Synopsis:

```
#include <math.h>

double ceil(double x);
float ceilf(float x);
long double ceill(long double x);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

cexp, cexpf, cexpl

Purpose: Complex exponential functions.

Synopsis:

```
#include <complex.h>

double complex cexp(double complex z);
float complex cexpf(float complex z);
long double complex cexpl(long double complex z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

cfgetispeed

Purpose: Get input baud rate.

Synopsis:

```
#include <termios.h>

speed_t cfgetispeed(const struct termios *termios_p);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

cfgetospeed

Purpose: Get output baud rate.

Synopsis: `#include <termios.h>`

```
speed_t cfgetospeed(const struct termios *termios_p);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

cfsetispeed

Purpose: Set input baud rate.

Synopsis: `#include <termios.h>`

```
int cfsetispeed(struct termios *termios_p, speed_t speed);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

cfsetospeed

Purpose: Set output baud rate.

Synopsis: `#include <termios.h>`

```
int cfsetospeed(struct termios *termios_p, speed_t speed);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

chdir

Purpose: Change working directory.

Synopsis: `#include <unistd.h>`

```
int chdir(const char *path);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than `{PATH_MAX}`.

chmod, fchmodat

Purpose: Change mode of a file relative to directory file descriptor.

Synopsis: `#include <sys/stat.h>`

```
int chmod(const char *path, mode_t mode);
```

```
int fchmodat(int fd, const char *path, mode_t mode, int flag);
```

The `fchmodat()` function is equivalent to the `chmod()` function except in the case where `path` specifies a relative path. In this case the file to be changed is determined relative to the directory associated with the file descriptor `fd` instead of the current working directory. If the file descriptor was opened without `O_SEARCH`, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with `O_SEARCH`, the function does not perform the check.

The `AT_SYMLINK_NOFOLLOW` flag can be used to specify that if *path* names a symbolic link, then the mode of the symbolic link is changed.

The purpose of the `fchmodat()` function is to enable changing the mode of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to `chmod()`, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the `fchmodat()` function it can be guaranteed that the changed file is located relative to the desired directory.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than `{PATH_MAX}`.

The `fchmodat()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

The `[ENOTDIR]` error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

chown, fchownat

Purpose: Change owner and group of a file relative to directory file descriptor.

Synopsis: `#include <unistd.h>`

```
int chown(const char *path, uid_t owner, gid_t group);
int fchownat(int fd, const char *path, uid_t owner,
             gid_t group, int flag);
```

The `fchownat()` function is equivalent to the `chown()` and `lchown()` functions except in the case where *path* specifies a relative path. In this case the file to be changed is determined relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without `O_SEARCH`, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with `O_SEARCH`, the function does not perform the check.

The `AT_SYMLINK_NOFOLLOW` flag controls whether `fchownat()` behaves like `chown()` or `lchown()`: if `AT_SYMLINK_NOFOLLOW` is set and *path* names a symbolic link, ownership of the symbolic link is changed.

The purpose of the `fchownat()` function is to enable changing ownership of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to `chown()` or `lchown()`, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the `fchownat()` function it can be guaranteed that the changed file is located relative to the desired directory.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than `{PATH_MAX}`.

The `fchownat()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

cimag, cimagf, cimagl

Purpose: Complex imaginary functions.

Synopsis: `#include <complex.h>`

```
double cimag(double complex z);
float cimagf(float complex z);
long double cimagl(long double complex z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

clearerr

Purpose: Clear indicators on a stream.

Synopsis: `#include <stdio.h>`

```
void clearerr(FILE *stream);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

clock

Purpose: Report CPU time used.

Synopsis: `#include <time.h>`

```
clock_t clock(void);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

clock_getcpuclockid

Purpose: Access a process CPU-time clock (**ADVANCED REALTIME**).

CPT

Synopsis: `#include <time.h>`

```
int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

clock_getres, clock_gettime, clock_settime

Purpose: Clock and timer functions.

cx Synopsis:

```
#include <time.h>

int clock_getres(clockid_t clock_id, struct timespec *res);
int clock_gettime(clockid_t clock_id, struct timespec *tp);
int clock_settime(clockid_t clock_id,
    const struct timespec *tp);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: Functionality relating to the Clock Selection option is moved to the Base. The *clock_getres()*, *clock_gettime()*, and *clock_settime()* functions are moved from the Timers option to the Base.

clock_nanosleep

Purpose: High resolution sleep with specifiable clock.

cx Synopsis:

```
#include <time.h>

int clock_nanosleep(clockid_t clock_id, int flags,
    const struct timespec *rqtp, struct timespec *rmtp);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The *clock_nanosleep()* function is moved from the Clock Selection option to the Base.

clog, clogf, clogl

Purpose: Complex natural logarithm functions.

Synopsis:

```
#include <complex.h>

double complex clog(double complex z);
float complex clogf(float complex z);
long double complex clogl(long double complex z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

close

Purpose: Close a file descriptor.

Synopsis:

```
#include <unistd.h>

int close(int fildes);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Functionality relating to the XSI STREAMS option is marked obsolescent. Functionality relating to the Asynchronous Input and Output and Memory Mapped Files options is moved to the Base.

Austin Group Interpretation 1003.1-2001 #139 is applied, clarifying that the requirement for `close()` on a socket to block for up to the current linger interval is not conditional on the `O_NONBLOCK` setting.

closedir

Purpose: Close a directory stream.

Synopsis: `#include <dirent.h>`
`int closedir(DIR *dirp);`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

closelog, openlog, setlogmask, syslog

Purpose: Control system log.

XSI Synopsis: `#include <syslog.h>`
`void closelog(void);`
`void openlog(const char *ident, int logopt, int facility);`
`int setlogmask(int maskpri);`
`void syslog(int priority, const char *message,`
`... /* arguments */);`

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

confstr

Purpose: Get configurable variables.

Synopsis: `#include <unistd.h>`
`size_t confstr(int name, char *buf, size_t len);`

Derivation: First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #047 is applied, adding the `_CS_V7_ENV` variable.

Austin Group Interpretations 1003.1-2001 #166 is applied to permit an additional compiler flag to enable threads.

The V6 variables for the supported programming environments are marked obsolescent.

The variables for the supported programming environments are updated to be V7.

The LEGACY variables and obsolescent values are removed.

conj, conjf, conjl

Purpose: Complex conjugate functions.

Synopsis:

```
#include <complex.h>

double complex conj(double complex z);
float complex conjf(float complex z);
long double complex conjl(long double complex z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

connect

Purpose: Connect a socket.

Synopsis:

```
#include <sys/socket.h>

int connect(int socket, const struct sockaddr *address,
            socklen_t address_len);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: Austin Group Interpretation 1003.1-2001 #035 is applied, clarifying the description of connected sockets.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

Austin Group Interpretation 1003.1-2001 #188 is applied, changing the method used to reset a peer address for a datagram socket.

copysign, copysignf, copysignl

Purpose: Number manipulation function.

Synopsis:

```
#include <math.h>

double copysign(double x, double y);
float copysignf(float x, float y);
long double copysignl(long double x, long double y);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

cos, cosf, cosl

Purpose: Cosine function.

Synopsis:

```
#include <math.h>

double cos(double x);
float cosf(float x);
long double cosl(long double x);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

cosh, coshf, coshl

Purpose: Hyperbolic cosine functions.

Synopsis: `#include <math.h>`
`double cosh(double x);`
`float coshf(float x);`
`long double coshl(long double x);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

cpow, cpowf, cpowl

Purpose: Complex power functions.

Synopsis: `#include <complex.h>`
`double complex cpow(double complex x, double complex y);`
`float complex cpowf(float complex x, float complex y);`
`long double complex cpowl(long double complex x,`
`long double complex y);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

cproj, cprojf, cprojl

Purpose: Complex projection functions.

Synopsis: `#include <complex.h>`
`double complex cproj(double complex z);`
`float complex cprojf(float complex z);`
`long double complex cprojl(long double complex z);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

creal, crealf, creall

Purpose: Complex real functions.

Synopsis: `#include <complex.h>`
`double creal(double complex z);`
`float crealf(float complex z);`
`long double creall(long double complex z);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

creat

Purpose: Create a new file or rewrite an existing one.

OH

Synopsis: `#include <sys/stat.h>`
`#include <fcntl.h>`

```
int creat(const char *path, mode_t mode);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

crypt

Purpose: String encoding function (**CRYPT**).

XSI

Synopsis: `#include <unistd.h>`
`char *crypt(const char *key, const char *salt);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: SD5-XSH-ERN-178 is applied, clarifying the required contents of the *salt* argument.

csin, csinf, csinl

Purpose: Complex sine functions.

Synopsis: `#include <complex.h>`

```
double complex csin(double complex z);
float complex csinf(float complex z);
long double complex csinl(long double complex z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

csinh, csinhf, csinhl

Purpose: Complex hyperbolic sine functions.

Synopsis: `#include <complex.h>`

```
double complex csinh(double complex z);
float complex csinhf(float complex z);
long double complex csinhl(long double complex z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

csqrt, csqrtf, csqrtl

Purpose: Complex square root functions.

Synopsis: `#include <complex.h>`

```
double complex csqrt(double complex z);
float complex csqrtf(float complex z);
long double complex csqrtl(long double complex z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

ctan, ctanf, ctanl

Purpose: Complex tangent functions.

Synopsis: `#include <complex.h>`

```
double complex ctan(double complex z);
float complex ctanf(float complex z);
long double complex ctanl(long double complex z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

ctanh, ctanhf, ctanhl

Purpose: Complex hyperbolic tangent functions.

Synopsis: `#include <complex.h>`

```
double complex ctanh(double complex z);
float complex ctanhf(float complex z);
long double complex ctanhl(long double complex z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

ctermid

Purpose: Generate a pathname for the controlling terminal.

cx Synopsis: `#include <stdio.h>`
`char *ctermid(char *s);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying the thread-safety requirements for the `ctermid()` function.

ctime, ctime_r

Purpose: Convert a time value to a date and time string.

OB Synopsis:

```
#include <time.h>
char *ctime(const time_t *clock);
OB CX char *ctime_r(const time_t *clock, char *buf);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: SD5-XSH-ERN-25 is applied, noting in APPLICATION USAGE that attempts to use *ctime()* or *ctime_r()* for times before the Epoch or for times beyond the year 9999 produce undefined results.

Austin Group Interpretation 1003.1-2001 #053 is applied, marking these functions obsolescent. Applications should use *strptime()* to generate strings from broken-down times. Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*.

The *ctime_r()* function is moved from the Thread-Safe Functions option to the Base.

dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey, dbm_nextkey, dbm_open, dbm_store

Purpose: Database functions.

XSI Synopsis:

```
#include <ndbm.h>
int dbm_clearerr(DBM *db);
void dbm_close(DBM *db);
int dbm_delete(DBM *db, datum key);
int dbm_error(DBM *db);
datum dbm_fetch(DBM *db, datum key);
datum dbm_firstkey(DBM *db);
datum dbm_nextkey(DBM *db);
DBM *dbm_open(const char *file, int open_flags,
mode_t file_mode);
int dbm_store(DBM *db, datum key, datum content,
int store_mode);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #042 is applied so that the DESCRIPTION permits newer implementations of the Berkeley DB interface.

difftime

Purpose: Compute the difference between two calendar time values.

Synopsis:

```
#include <time.h>
double difftime(time_t time1, time_t time0);
```

Derivation: First released in Issue 4. Derived from the ISO C standard.

Issue 7: No functional changes are made in this issue.

dirfd

Purpose: Extract the file descriptor used by a DIR stream.

Synopsis:

```
#include <dirent.h>
int dirfd(DIR *dirp);
```

The *dirfd()* function returns a file descriptor referring to the same directory as the *dirp* argument. This file descriptor is closed by a call to *closedir()*.

The *dirfd()* function is intended to be a mechanism by which an application may obtain a file descriptor to use for the *fchdir()* function.

This interface was introduced because the Base Definitions volume of POSIX.1-2008 does not make public the **DIR** data structure. Applications tend to use the *fchdir()* function on the file descriptor returned by this interface, and this has proven useful for security reasons; in particular, it is a better technique than others where directory names might change.

The description uses the term “a file descriptor” rather than “the file descriptor”. The implication intended is that an implementation that does not use an *fd* for *diropen()* could still *open()* the directory to implement the *dirfd()* function. Such a descriptor must be closed later during a call to *closedir()*.

An implementation that does not support file descriptors referring to directories may fail with [ENOTSUP].

If it is necessary to allocate an *fd* to be returned by *dirfd()*, it should be done at the time of a call to *opendir()*.

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Issue 7: First released in Issue 7.

dirname

Purpose: Report the parent directory name of a file pathname.

XS1 Synopsis:

```
#include <libgen.h>
char *dirname(char *path);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The EXAMPLES section is revised.

div

Purpose: Compute the quotient and remainder of an integer division.

Synopsis:

```
#include <stdlib.h>
div_t div(int numer, int denom);
```

Derivation: First released in Issue 4. Derived from the ISO C standard.

Issue 7: No functional changes are made in this issue.

dlclose

Purpose: Close a *dlopen()* object.

Synopsis:

```
#include <dlfcn.h>
int dlclosel(void *handle);
```

Derivation: First released in Issue 5.

Issue 7: The *dlopen()* function is moved from the XSI option to Base.

dlderror

Purpose: Get diagnostic information.

Synopsis:

```
#include <dlfcn.h>
char *dlderror(void);
```

Derivation: First released in Issue 5.

Issue 7: The *dlderror()* function is moved from the XSI option to the Base.

dlopen

Purpose: Gain access to an executable object file.

Synopsis:

```
#include <dlfcn.h>
void *dlopen(const char *file, int mode);
```

Derivation: First released in Issue 5.

Issue 7: The *dlopen()* function is moved from the XSI option to the Base.
The EXAMPLES section is updated to refer to *dlsym()*.

dlsym

Purpose: Obtain the address of a symbol from a *dlopen()* object.

Synopsis:

```
#include <dlfcn.h>
void *dlsym(void *restrict handle, const char *restrict name);
```

Derivation: First released in Issue 5.

Issue 7: The *dlsym()* function is moved from the XSI option to the Base.

drand48, erand48, jrand48, lcong48, lrand48, mrand48, nrand48, seed48, srand48

Purpose: Generate uniformly distributed pseudo-random numbers.

XSI Synopsis:

```
#include <stdlib.h>
double drand48(void);
double erand48(unsigned short xsubi[3]);
long jrand48(unsigned short xsubi[3]);
void lcong48(unsigned short param[7]);
long lrand48(void);
long mrand48(void);
long nrand48(unsigned short xsubi[3]);
```

```
unsigned short *seed48(unsigned short seed16v[3]);
void srand48(long seedval);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

dup, dup2

Purpose: Duplicate an open file descriptor.

Synopsis: `#include <unistd.h>`
`int dup(int fildes);`
`int dup2(int fildes, int fildes2);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: SD5-XSH-ERN-187 is applied, clarifying several aspects of the behavior of `dup2()`.

duplocale

Purpose: Duplicate a locale object.

cx Synopsis: `#include <locale.h>`
`locale_t duplocale(locale_t locobj);`

The `duplocale()` function creates duplicate copy of the locale object referenced by the `locobj` argument.

The use of the `duplocale()` function is recommended for situations where a locale object is being used in multiple places, and it is possible that the lifetime of the locale object might end before all uses are finished. Another reason to duplicate a locale object is if a slightly modified form is needed. This can be achieved by a call to `newlocale()` following the `duplocale()` call.

As with the `newlocale()` function, handles for locale objects created by the `duplocale()` function should be released by a corresponding call to `freelocale()`.

The following example shows a code fragment to create a slightly altered version of an existing locale object. The function takes a locale object and a locale name and it replaces the `LC_TIME` category data in the locale object with that from the named locale.

```
#include <locale.h>
...
locale_t
with_changed_lc_time (locale_t obj, const char *name)
{
    locale_t retval = duplocale (obj);
    if (retval != (locale_t) 0)
    {
        locale_t changed = newlocale (LC_TIME_MASK, name,
            retval);
        if (changed == (locale_t) 0)
            /* An error occurred. Free all allocated resources. */
    }
}
```

```

        freelocale (retval);
        retval = changed;
    }
    return retval; }
}

```

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 4.

Issue 7: First released in Issue 7.

encrypt

Purpose: Encoding function (**CRYPT**).

XSI Synopsis:

```
#include <unistd.h>
void encrypt(char block[64], int edflag);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

endgrent, getgrent, setgrent

Purpose: Group database entry functions.

XSI Synopsis:

```
#include <grp.h>
void endgrent(void);
struct group *getgrent(void);
void setgrent(void);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

endhostent, gethostent, sethostent

Purpose: Network host database functions.

Synopsis:

```
#include <netdb.h>
void endhostent(void);
struct hostent *gethostent(void);
void sethostent(int stayopen);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

endnetent, getnetbyaddr, getnetbyname, getnetent, setnetent

Purpose: Network database functions.

Synopsis: `#include <netdb.h>`

```
void endnetent(void);
struct netent *getnetbyaddr(uint32_t net, int type);
struct netent *getnetbyname(const char *name);
struct netent *getnetent(void);
void setnetent(int stayopen);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

endprotoent, getprotobyname, getprotobynumber, getprotoent, setprotoent

Purpose: Network protocol database functions.

Synopsis: `#include <netdb.h>`

```
void endprotoent(void);
struct protoent *getprotobyname(const char *name);
struct protoent *getprotobynumber(int proto);
struct protoent *getprotoent(void);
void setprotoent(int stayopen);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

endpwent, getpwent, setpwent

Purpose: User database functions.

XS1 Synopsis: `#include <pwd.h>`

```
void endpwent(void);
struct passwd *getpwent(void);
void setpwent(void);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.
The EXAMPLES section is revised.

endservent, getservbyname, getservbyport, getservent, setservent

Purpose: Network services database functions.

Synopsis: `#include <netdb.h>`

```
void endservent(void);
struct servent *getservbyname(const char *name,
                             const char *proto);
struct servent *getservbyport(int port, const char *proto);
struct servent *getservent(void);
void setservent(int stayopen);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: SD5-XBD-ERN-14 is applied, clarifying the way in which port numbers are converted to and from network byte order.

endutxent, getutxent, getutxid, getutxline, pututxline, setutxent

Purpose: User accounting database functions.

XS1 Synopsis:

```
#include <utmpx.h>

void endutxent(void);
struct utmpx *getutxent(void);
struct utmpx *getutxid(const struct utmpx *id);
struct utmpx *getutxline(const struct utmpx *line);
struct utmpx *pututxline(const struct utmpx *utmpx);
void setutxent(void);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

erf, erff, erfl

Purpose: Error functions.

Synopsis:

```
#include <math.h>

double erf(double x);
float erff(float x);
long double erfl(long double x);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

erfc, erfcf, erfcl

Purpose: Complementary error functions.

Synopsis:

```
#include <math.h>

double erfc(double x);
float erfcf(float x);
long double erfcl(long double x);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

errno

Purpose: Error return value.

Synopsis:

```
#include <errno.h>
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

environ, execl, execl_e, execlp, execv, execve, execvp, fexecve

Purpose: Execute a file.

Synopsis: `#include <unistd.h>`

```
extern char **environ;
int execl(const char *path, const char *arg0,
... /*, (char *)0 */);
int execl_e(const char *path, const char *arg0, ... /*,
(char *)0, char *const envp[]*/);
int execlp(const char *file, const char *arg0, ... /*,
(char *)0 */);
int execv(const char *path, char *const argv[]);
int execve(const char *path, char *const argv[],
char *const envp[]);
int execvp(const char *file, char *const argv[]);
int fexecve(int fd, char *const argv[], char *const envp[]);
```

The `fexecve()` function is equivalent to the `execve()` function except that the file to be executed is determined by the file descriptor `fd` instead of a pathname. The file offset of `fd` is ignored.

The purpose of the `fexecve()` function is to enable executing a file which has been verified to be the intended file. It is possible to actively check the file by reading from the file descriptor and be sure that the file is not exchanged for another between the reading and the execution. Alternatively, an function like `openat()` can be used to open a file which has been found by reading the content of a directory using `readdir()`.

Since execute permission is checked by `fexecve()`, the file descriptor `fd` need not have been opened with the `O_EXEC` flag. However, if the file to be executed denies read and write permission for the process preparing to do the `exec`, the only way to provide the `fd` to `fexecve()` will be to use the `O_EXEC` flag when opening `fd`. In this case, the application will not be able to perform a checksum test since it will not be able to read the contents of the file.

Note that when a file descriptor is opened with `O_RDONLY`, `O_RDWR`, or `O_WRONLY` mode, the file descriptor can be used to read, read and write, or write the file, respectively, even if the mode of the file changes after the file was opened. Using the `O_EXEC` open mode is different; `fexecve()` will ignore the mode that was used when the file descriptor was opened and the `exec` will fail if the mode of the file associated with `fd` does not grant execute permission to the calling process at the time `fexecve()` is called.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #047 is applied, adding a warning for `execl_e()`, `execve()` and `fexecve()` to the APPLICATION USAGE that the new process might be invoked in a nonconforming environment if the `envp` array does not contain implementation-defined variables required by the implementation to provide a conforming environment. See the `_CS_V7_ENV` entry in `<unistd.h>`, and `confstr()`, for details.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than `{PATH_MAX}`.

The `fexecve()` function is added from The Open Group Technical Standard, 2006,

Extended API Set Part 2.

Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, Threads, and Timers options is moved to the Base.

Changes are made related to support for finegrained timestamps.

exit

Purpose: Terminate a process.

Synopsis: `#include <stdlib.h>`
`void exit(int status);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #085 is applied, deleting the reference to removal of files created by `tmpfile()`.

exp, expf, expl

Purpose: Exponential function.

Synopsis: `#include <math.h>`
`double exp(double x);`
`float expf(float x);`
`long double expl(long double x);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

exp2, exp2f, exp2l

Purpose: Exponential base 2 functions.

Synopsis: `#include <math.h>`
`double exp2(double x);`
`float exp2f(float x);`
`long double exp2l(long double x);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

expm1, expm1f, expm1l

Purpose: Compute exponential functions.

Synopsis: `#include <math.h>`
`double expm1(double x);`
`float expm1f(float x);`
`long double expm1l(long double x);`

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

fabs, fabsf, fabsl

Purpose: Absolute value function.

Synopsis: `#include <math.h>`
`double fabs(double x);`
`float fabsf(float x);`
`long double fabsl(long double x);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

fattach

Purpose: Attach a STREAMS-based file descriptor to a file in the file system name space (**STREAMS**).

OB XSR Synopsis: `#include <stropts.h>`
`int fattach(int fildev, const char *path);`

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *fattach*() function is marked obsolescent.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

fchdir

Purpose: Change working directory.

Synopsis: `#include <unistd.h>`
`int fchdir(int fildev);`

Derivation: First released in Issue 4, Version 2.

Issue 7: The *fchdir*() function is moved from the XSI option to the Base.

fchmod

Purpose: Change mode of a file.

Synopsis: `#include <sys/stat.h>`
`int fchmod(int fildev, mode_t mode);`

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

fchown

Purpose: Change owner and group of a file.

Synopsis: `#include <unistd.h>`
`int fchown(int fildes, uid_t owner, gid_t group);`

Derivation: First released in Issue 4, Version 2.

Issue 7: Functionality relating to XSI STREAMS is marked obsolescent.

fclose

Purpose: Close a stream.

Synopsis: `#include <stdio.h>`
`int fclose(FILE *stream);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #002 is applied, clarifying the interaction of file descriptors and streams.

The [ENOSPC] error condition is updated and the [ENOMEM] error is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Changes are made related to support for finegrained timestamps.

fcntl

Purpose: File control.

Synopsis: `#include <fcntl.h>`
`int fcntl(int fildes, int cmd, ...);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #150 is applied, clarifying the file status flags returned when *cmd* is F_GETFL.

Austin Group Interpretation 1003.1-2001 #171 is applied, adding the F_DUPFD_CLOEXEC flag.

The optional **<unistd.h>** header is removed from this function, since **<fcntl.h>** now defines SEEK_SET, SEEK_CUR, and SEEK_END as part of the Base.

fdatasync

Purpose: Synchronize the data of a file (**REALTIME**).

sio Synopsis: `#include <unistd.h>`
`int fdatasync(int fildes);`

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: No functional changes are made in this issue.

fdetach

Purpose: Detach a name from a STREAMS-based file descriptor (f3STREAMSfP).

OB XSR Synopsis:

```
#include <stropts.h>
int fdetach(const char *path);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *fdetach()* function is marked obsolescent.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

fdim, fdimf, fdiml

Purpose: Compute positive difference between two floating-point numbers.

Synopsis:

```
#include <math.h>
double fdim(double x, double y);
float fdimf(float x, float y);
long double fdiml(long double x, long double y);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

fdopen

Purpose: Associate a stream with a file descriptor.

CX Synopsis:

```
#include <stdio.h>
FILE *fdopen(int fildes, const char *mode);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: SD5-XSH-ERN-149 is applied, adding the {STREAM_MAX} [EMFILE] error condition.

Changes are made related to support for finegrained timestamps.

fdopendir, opendir

Purpose: Open directory associated with file descriptor.

Synopsis:

```
#include <dirent.h>
DIR *fdopendir(int fd);
DIR *opendir(const char *dirname);
```

The *fdopendir()* function is equivalent to the *opendir()* function except that the directory is specified by a file descriptor rather than by a name. The file offset associated with the file descriptor at the time of the call determines which entries are returned.

Upon successful return from `fdopendir()`, the file descriptor is under the control of the system, and if any attempt is made to close the file descriptor, or to modify the state of the associated description, other than by means of `closedir()`, `readdir()`, `readdir_r()`, or `rewinddir()`, the behavior is undefined. Upon calling `closedir()` the file descriptor is closed.

The purpose of the `fdopendir()` function is to enable opening files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to `opendir()`, resulting in unspecified behavior.

The following example program searches through a given directory looking for files whose name does not begin with a dot and whose size is larger than 1 MiB.

```
#include <stdio.h>
#include <dirent.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdint.h>
#include <stdlib.h>
#include <unistd.h>

int
main(int argc, char *argv[])
{
    struct stat statbuf;
    DIR *d;
    struct dirent *dp;
    int dfd, ffd;

    if ((d = fdopendir((dfd = open("./tmp", O_RDONLY))))
        == NULL) {
        fprintf(stderr, "Cannot open ./tmp directory\n");
        exit(1);
    }
    while ((dp = readdir(d)) != NULL) {
        if (dp->d_name[0] == '.')
            continue;
        /* there is a possible race condition here as the file
         * could be renamed between the readdir and the open */
        if ((ffd = openat(dfd, dp->d_name, O_RDONLY)) == -1) {
            perror(dp->d_name);
            continue;
        }
        if (fstat(ffd, &statbuf) == 0 && statbuf.st_size >
            (1024*1024)) {
            /* found it ... */
            printf("%s: %jdK\n", dp->d_name,
                (intmax_t)(statbuf.st_size / 1024));
        }
        close(ffd);
    }
    closedir(d); // note this implicitly closes dfd
    return 0;
}
```


Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

The *fdopendir()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

An additional example is added.

feclearexcept

Purpose: Clear floating-point exception.

Synopsis:

```
#include <fenv.h>

int feclearexcept(int excepts);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

fegetenv, fesetenv

Purpose: Get and set current floating-point environment.

Synopsis:

```
#include <fenv.h>

int fegetenv(fenv_t *envp);
int fesetenv(const fenv_t *envp);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

fegetexceptflag, fesetexceptflag

Purpose: Get and set floating-point status flags.

Synopsis:

```
#include <fenv.h>

int fegetexceptflag(fexcept_t *flagp, int excepts);
int fesetexceptflag(const fexcept_t *flagp, int excepts);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

fegetround, fesetround

Purpose: Get and set current rounding direction.

Synopsis:

```
#include <fenv.h>

int fegetround(void);
int fesetround(int round);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

feholdexcept

Purpose: Save current floating-point environment.

Synopsis:

```
#include <fenv.h>
int feholdexcept(fenv_t *envp);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

feof

Purpose: Test end-of-file indicator on a stream.

Synopsis:

```
#include <stdio.h>
int feof(FILE *stream);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

feraiseexcept

Purpose: Raise floating-point exception.

Synopsis:

```
#include <fenv.h>
int feraiseexcept(int excepts);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

ferror

Purpose: Test error indicator on a stream.

Synopsis:

```
#include <stdio.h>
int ferror(FILE *stream);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

fetestexcept

Purpose: Test floating-point exception flags.

Synopsis:

```
#include <fenv.h>
int fetestexcept(int excepts);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

feupdateenv

Purpose: Update floating-point environment.

Synopsis: `#include <fenv.h>`
`int feupdateenv(const fenv_t *envp);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

fflush

Purpose: Flush a stream.

Synopsis: `#include <stdio.h>`
`int fflush(FILE *stream);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #002 is applied, clarifying the interaction of file descriptors and streams.

The [ENOSPC] error condition is updated and the [ENOMEM] error is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The EXAMPLES section is revised.

Changes are made related to support for finegrained timestamps.

ffs

Purpose: Find first set bit.

XSI Synopsis: `#include <strings.h>`
`int ffs(int i);`

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

fgetc

Purpose: Get a byte from a stream.

Synopsis: `#include <stdio.h>`
`int fgetc(FILE *stream);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #051 is applied, updating the list of functions that mark the last data access timestamp for update.

fgetpos

Purpose: Get current file position information.

Synopsis: `#include <stdio.h>`
`int fgetpos(FILE *restrict stream, fpos_t *restrict pos);`

Derivation: First released in Issue 4. Derived from the ISO C standard.

Issue 7: No functional changes are made in this issue.

fgets

Purpose: Get a string from a stream.

Synopsis: `#include <stdio.h>`
`char *fgets(char *restrict s, int n, FILE *restrict stream);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #051 is applied, updating the list of functions that mark the last data access timestamp for update.

fgetwc

Purpose: Get a wide-character code from a stream.

Synopsis: `#include <stdio.h>`
`#include <wchar.h>`
`wint_t fgetwc(FILE *stream);`

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.

Changes are made related to support for finegrained timestamps.

fgetws

Purpose: Get a wide-character string from a stream.

Synopsis: `#include <stdio.h>`
`#include <wchar.h>`
`wchar_t *fgetws(wchar_t *restrict ws, int n,`
`FILE *restrict stream);`

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.

Changes are made related to support for finegrained timestamps.

fileno

Purpose: Map a stream pointer to a file descriptor.

cx Synopsis:

```
#include <stdio.h>
int fileno(FILE *stream);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: SD5-XBD-ERN-99 is applied, changing the definition of the [EBADF] error.

flockfile, ftrylockfile, funlockfile

Purpose: Stdio locking functions.

cx Synopsis:

```
#include <stdio.h>
void flockfile(FILE *file);
int ftrylockfile(FILE *file);
void funlockfile(FILE *file);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The *flockfile()*, *ftrylockfile()*, and *funlockfile()* functions are moved from the Thread-Safe Functions option to the Base.

floor, floorf, floorl

Purpose: Floor function.

Synopsis:

```
#include <math.h>
double floor(double x);
float floorf(float x);
long double floorl(long double x);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

fma, fmaf, fmal

Purpose: Floating-point multiply-add.

Synopsis:

```
#include <math.h>
double fma(double x, double y, double z);
float fmaf(float x, float y, float z);
long double fmal(long double x, long double y, long double z);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #57 (SD5-XSH-ERN-69) is applied, adding a “may fail” range error for non-MX systems.

fmax, fmaxf, fmaxl

Purpose: Determine maximum numeric value of two floating-point numbers.

Synopsis: `#include <math.h>`
`double fmax(double x, double y);`
`float fmaxf(float x, float y);`
`long double fmaxl(long double x, long double y);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #007 is applied, adding MX shading where the text refers to NaNs.

fmemopen

Purpose: Open a memory buffer stream.

cx Synopsis: `#include <stdio.h>`
`FILE *fmemopen(void *restrict buf, size_t size,`
`const char *restrict mode);`

The *fmemopen()* function associates the buffer given by the *buf* and *size* arguments with a stream.

This interface has been introduced to eliminate many of the errors encountered in the construction of strings, notably overflowing of strings. This interface prevents overflow.

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Issue 7: First released in Issue 7.

fmin, fminf, fminl

Purpose: Determine minimum numeric value of two floating-point numbers.

Synopsis: `#include <math.h>`
`double fmin(double x, double y);`
`float fminf(float x, float y);`
`long double fminl(long double x, long double y);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #008 is applied, adding MX shading where the text refers to NaNs.

fmod, fmodf, fmodl

Purpose: Floating-point remainder value function.

Synopsis: `#include <math.h>`
`double fmod(double x, double y);`
`float fmodf(float x, float y);`
`long double fmodl(long double x, long double y);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

fmtmsg

Purpose: Display a message in the specified format on standard error and/or a system console.

xSI Synopsis:

```
#include <fmtmsg.h>

int fmtmsg(long classification, const char *label,
            int severity, const char *text,
            const char *action, const char *tag);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

fnmatch

Purpose: Match a filename or a pathname.

Synopsis:

```
#include <fnmatch.h>

int fnmatch(const char *pattern, const char *string,
            int flags);
```

Derivation: First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 7: No functional changes are made in this issue.

fopen

Purpose: Open a stream.

Synopsis:

```
#include <stdio.h>

FILE *fopen(const char *restrict filename,
            const char *restrict mode);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #025 is applied, clarifying the file creation mode.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

Austin Group Interpretation 1003.1-2001 #159 is applied, clarifying requirements for the flags set on the open file description.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-149 is applied, changing the {STREAM_MAX} [EMFILE] error condition from a “may fail” to a “shall fail”.

Changes are made related to support for finegrained timestamps.

fork

Purpose: Create a new process.

Synopsis: `#include <unistd.h>`
`pid_t fork(void);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #080 is applied, clarifying the status of asynchronous input and asynchronous output operations and asynchronous control lists in the DESCRIPTION.

Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, Timers, and Threads options is moved to the Base.

Functionality relating to message catalog descriptors is moved from the XSI option to the Base.

fpathconf, pathconf

Purpose: Get configurable pathname variables.

Synopsis: `#include <unistd.h>`
`long fpathconf(int fildev, int name);`
`long pathconf(const char *path, int name);`

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

Austin Group Interpretation 1003.1-2001 #160 is applied, clarifying the requirements related to variables that have no limit.

Changes are made related to support for finegrained timestamps.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

fpclassify

Purpose: Classify real floating type.

Synopsis: `#include <math.h>`
`int fpclassify(real-floating x);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

dprintf, fprintf, printf, snprintf, sprintf

Purpose: Print formatted output.

Synopsis: #include <stdio.h>

```

cx      int dprintf(int fildev, const char *restrict format, ...);
        int fprintf(FILE *restrict stream,
            const char *restrict format, ...);
        int printf(const char *restrict format, ...);
        int snprintf(char *restrict s, size_t n,
            const char *restrict format, ...);
        int sprintf(char *restrict s,
            const char *restrict format, ...);

```

The *dprintf()* function is equivalent to the *fprintf()* function, except that *dprintf()* writes output to the file associated with the file descriptor specified by the *fildev* argument rather than placing output on a stream.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #161 is applied, updating the DESCRIPTION of the 0 flag.

Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a “may fail” to a “shall fail”.

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #68 (SD5-XSH-ERN-70) is applied, revising the description of *g* and *G*.

The *dprintf()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Functionality relating to the *%n\$* form of conversion specification and the <apostrophe> flag is moved from the XSI option to the Base.

Changes are made related to support for finegrained timestamps.

fputc

Purpose: Put a byte on a stream.

Synopsis: #include <stdio.h>

```
int fputc(int c, FILE *stream);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Changes are made related to support for finegrained timestamps.

fputs

Purpose: Put a string on a stream.

Synopsis: #include <stdio.h>

```
int fputs(const char *restrict s, FILE *restrict stream);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Changes are made related to support for finegrained timestamps.

fputwc

Purpose: Put a wide-character code on a stream.

Synopsis:

```
#include <stdio.h>
#include <wchar.h>

wint_t fputwc(wchar_t wc, FILE *stream);
```

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: Changes are made related to support for finegrained timestamps.

fputws

Purpose: Put a wide-character string on a stream.

Synopsis:

```
#include <stdio.h>
#include <wchar.h>

int fputws(const wchar_t *restrict ws, FILE *restrict stream);
```

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: Changes are made related to support for finegrained timestamps.

fread

Purpose: Binary input.

Synopsis:

```
#include <stdio.h>

size_t fread(void *restrict ptr, size_t size, size_t nitems,
FILE *restrict stream);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Changes are made related to support for finegrained timestamps.

free

Purpose: Free allocated memory.

Synopsis:

```
#include <stdlib.h>

void free(void *ptr);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The DESCRIPTION is updated to clarify that if the pointer returned is not by a function that allocates memory as if by *malloc()*, then the behavior is undefined.

freeaddrinfo, getaddrinfo

Purpose: Get address information.

Synopsis:

```
#include <sys/socket.h>
#include <netdb.h>

void freeaddrinfo(struct addrinfo *ai);
int getaddrinfo(const char *restrict nodename,
const char *restrict servname,
const struct addrinfo *restrict hints,
struct addrinfo **restrict res);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: Austin Group Interpretation 1003.1-2001 #013 is applied, removing the [EAI_OVERFLOW] error code.

Austin Group Interpretation 1003.1-2001 #146 is applied, eliminating the use of “may” in relation to the *hints* argument.

An example is added.

freelocale

Purpose: Free resources allocated for a locale object.

cx

Synopsis:

```
#include <locale.h>
void freelocale(locale_t locobj);
```

The *freelocale()* function causes the resources allocated for a locale object returned by a call to the *newlocale()* or *duplocale()* functions to be released.

The following example shows a code fragment to free a locale object created by *newlocale()*:

```
#include <locale.h>
...
/* Every locale object allocated with newlocale() should be
 * freed using freelocale():
 */
locale_t loc;
/* Get the locale. */
loc = newlocale (LC_CTYPE_MASK | LC_TIME_MASK, "locname", NULL);
/* ... Use the locale object ... */
...
/* Free the locale object resources. */
freelocale (loc);
```

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 4.

Issue 7: First released in Issue 7.

freopen

Purpose: Open a stream.

Synopsis:

```
#include <stdio.h>
FILE *freopen(const char *restrict filename,
              const char *restrict mode, FILE *restrict stream);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #043 is applied, clarifying that the *freopen()* function allocates a file descriptor as per *open()*.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations

to support pathnames longer than {PATH_MAX}.

Austin Group Interpretation 1003.1-2001 #159 is applied, clarifying requirements for the flags set on the open file description.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-150 is applied, clarifying the DESCRIPTION.

SD5-XSH-ERN-219 is applied, adding advice to the APPLICATION USAGE relating to the use of a NULL *filename* argument.

frexp, frexpf, frexpl

Purpose: Extract mantissa and exponent from a double precision number.

Synopsis: `#include <math.h>`

```
double frexp(double num, int *exp);
float frexpf(float num, int *exp);
long double frexpl(long double num, int *exp);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

fscanf, scanf, sscanf

Purpose: Convert formatted input.

Synopsis: `#include <stdio.h>`

```
int fscanf(FILE *restrict stream,
           const char *restrict format, ...);
int scanf(const char *restrict format, ...);
int sscanf(const char *restrict s,
           const char *restrict format, ...);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a “may fail” to a “shall fail”.

SD5-XSH-ERN-9 is applied, correcting *fscanf()* to *scanf()* in the DESCRIPTION.

SD5-XSH-ERN-132 is applied, adding the assignment-allocation character ‘m’.

Functionality relating to the %n\$ form of conversion specification is moved from the XSI option to the Base.

Changes are made related to support for finegrained timestamps.

fseek, fseeko

Purpose: Reposition a file-position indicator in a stream.

Synopsis: `#include <stdio.h>`

```
int fseek(FILE *stream, long offset, int whence);
int fseeko(FILE *stream, off_t offset, int whence);
```

CX

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Changes are made related to support for finegrained timestamps.

fsetpos

Purpose: Set current file position.

Synopsis: `#include <stdio.h>`

```
int fsetpos(FILE *stream, const fpos_t *pos);
```

Derivation: First released in Issue 4. Derived from the ISO C standard.

Issue 7: SD5-XSH-ERN-220 is applied, changing the first [EPIPE] to [ESPIPE].

fstat

Purpose: Get file status.

Synopsis: `#include <sys/stat.h>`

```
int fstat(int fildes, struct stat *buf);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: XSH-SD5-ERN-161 is applied, updating the DESCRIPTION to clarify to which file types *st_nlink* applies.

Changes are made related to support for finegrained timestamps.

fstatat, lstat, stat

Purpose: Get file status.

Synopsis: `#include <sys/stat.h>`

```
int fstatat(int fd, const char *restrict path,
            struct stat *restrict buf, int flag);
int lstat(const char *restrict path,
          struct stat *restrict buf);
int stat(const char *restrict path,
         struct stat *restrict buf);
```

The *fstatat()* function is equivalent to the *stat()* and *lstat()* functions, except in the case where *path* specifies a relative path. In this case the status is retrieved from a file relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without `O_SEARCH`, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with `O_SEARCH`, the function does not perform the check.

The `AT_SYMLINK_NOFOLLOW` flag controls whether *fchownat()* behaves like *stat()* or *lstat()*: if `AT_SYMLINK_NOFOLLOW` is set and *path* names a symbolic link, the status of the symbolic link is returned.

The purpose of the *fstatat()* function is to obtain the status of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *stat()* or *lstat()*, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *fstatat()* function it can be guaranteed that the file for which status is returned is located relative to the desired directory.

- Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.
- Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.
- XSH-SD5-ERN-161 is applied, updating the DESCRIPTION to clarify to which file types *st_nlink* applies.
- The *fstatat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.
- Changes are made related to support for finegrained timestamps.
- The *lstat()* function is now required to return meaningful data for symbolic links in all **stat** structure fields, except for the permission bits of *st_mode*.
- The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

fstatvfs, statvfs

Purpose: Get file system information.

Synopsis:

```
#include <sys/statvfs.h>

int fstatvfs(int fildev, struct statvfs *buf);
int statvfs(const char *restrict path,
            struct statvfs *restrict buf);
```

Derivation: First released in Issue 4, Version 2.

- Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.
- SD5-XSH-ERN-68 is applied, correcting the EXAMPLES section.
- The *fstatvfs()* and *statvfs()* functions are moved from the XSI option to the Base.
- The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

fsync

Purpose: Synchronize changes to a file.

fsc Synopsis:

```
#include <unistd.h>
int fsync(int fildev);
```

Derivation: First released in Issue 3.

- Issue 7: No functional changes are made in this issue.

ftell, ftello

Purpose: Return a file offset in a stream.

Synopsis: `#include <stdio.h>`

CX `long ftell(FILE *stream);`
`off_t ftello(FILE *stream);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

ftok

Purpose: Generate an IPC key.

XSI `#include <sys/ipc.h>`
`key_t ftok(const char *path, int id);`

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

ftruncate

Purpose: Truncate a file to a specified length.

Synopsis: `#include <unistd.h>`

`int ftruncate(int fildes, off_t length);`

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #056 is applied, revising the ERRORS section (although the [EINVAL] “may fail” error was subsequently removed during review of the XSI option).

Functionality relating to the Memory Protection and Memory Mapped Files options is moved to the Base.

The DESCRIPTION is updated so that a call to *ftruncate()* when the file is smaller than the size requested will increase the size of the file. Previously, non-XSI-conforming implementations were allowed to increase the size of the file or fail.

Changes are made related to support for finegrained timestamps.

ftw

Purpose: Traverse (walk) a file tree.

OB XSI Synopsis:

```
#include <ftw.h>
int ftw(const char *path, int (*fn)(const char *,
    const struct stat *ptr, int flag), int ndirs);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *ftw()* function is marked obsolescent. Applications should use the *nftw()* function instead.

futimens, utimensat, utimes

Purpose: Set file access and modification times.

Synopsis:

```
#include <sys/stat.h>
int futimens(int fd, const struct timespec times[2]);
int utimensat(int fd, const char *path,
    const struct timespec times[2], int flag);
```

XSI

```
#include <sys/time.h>
int utimes(const char *path, const struct timeval times[2]);
```

The *futimens()* and *utimensat()* functions set the access and modification times of a file to the values of the *times* argument. The *futimens()* function changes the times of the file associated with the file descriptor *fd*. The *utimensat()* function changes the times of the file pointed to by the *path* argument, relative to the directory associated with the file descriptor *fd*.

The *times* argument is an array of two **timespec** structures. The first array member represents the date and time of last access, and the second member represents the date and time of last modification. The times in the **timespec** structure are measured in seconds and nanoseconds since the Epoch. The file's relevant timestamp is set to the greatest value supported by the file system that is not greater than the specified time.

If the *tv_nsec* field of a **timespec** structure has the special value `UTIME_NOW`, the file's relevant timestamp is set to the greatest value supported by the file system that is not greater than the current time. If the *tv_nsec* field has the special value `UTIME_OMIT`, the file's relevant timestamp is not changed. In either case, the *tv_sec* field is ignored.

If *utimensat()* is passed a relative path in the *path* argument, the file to be used is relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without `O_SEARCH`, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with `O_SEARCH`, the function does not perform the check.

The `AT_SYMLINK_NOFOLLOW` flag can be used to specify that if *path* names a symbolic link, then the access and modification times of the symbolic link are

changed.

The purpose of the `utimensat()` function is to set the access and modification time of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to `utimes()`, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the `utimensat()` function it can be guaranteed that the changed file is located relative to the desired directory.

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than `{PATH_MAX}`.

The LEGACY marking is removed from `utimes()`.

The `utimensat()` function (renamed from `futimesat()`) is added from The Open Group Technical Standard, 2006, Extended API Set Part 2, and changed to allow modifying a symbolic link by adding a `flag` argument.

The `futimens()` function is added.

Changes are made related to support for finegrained timestamps.

The `[ENOTDIR]` error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

fwide

Purpose: Set stream orientation.

Synopsis:

```
#include <stdio.h>
#include <wchar.h>

int fwide(FILE *stream, int mode);
```

Derivation: First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: No functional changes are made in this issue.

fwprintf, swprintf, wprintf

Purpose: Print formatted wide-character output.

Synopsis:

```
#include <stdio.h>
#include <wchar.h>

int fwprintf(FILE *restrict stream,
             const wchar_t *restrict format, ...);
int swprintf(wchar_t *restrict ws, size_t n,
             const wchar_t *restrict format, ...);
int wprintf(const wchar_t *restrict format, ...);
```

Derivation: First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: Austin Group Interpretation 1003.1-2001 #161 is applied, updating the DESCRIPTION of the `0` flag.

Austin Group Interpretation 1003.1-2001 #170 is applied, changing the `[EILSEQ]`

error condition from a “may fail” to a “shall fail”.

ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #68 (SD5-XSH-ERN-70) is applied, revising the description of `g` and `G`.

Functionality relating to the “%n\$” form of conversion specification and the <apostrophe> flag is moved from the XSI option to the Base.

The [EOverflow] error is added for `swprintf()`.

Changes are made related to support for finegrained timestamps.

fwrite

Purpose: Binary output.

Synopsis:

```
#include <stdio.h>

size_t fwrite(const void *restrict ptr, size_t size,
              size_t nitems, FILE *restrict stream);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Changes are made related to support for finegrained timestamps.

fwscanf, swscanf, wscanf

Purpose: Convert formatted wide-character input.

Synopsis:

```
#include <stdio.h>
#include <wchar.h>

int fwscanf(FILE *restrict stream,
            const wchar_t *restrict format, ...);
int swscanf(const wchar_t *restrict ws,
            const wchar_t *restrict format, ...);
int wscanf(const wchar_t *restrict format, ...);
```

Derivation: First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a “may fail” to a “shall fail”.

SD5-XSH-ERN-132 is applied, adding the assignment-allocation character ‘m’.

Functionality relating to the “%n\$” form of conversion specification is moved from the XSI option to the Base.

Changes are made related to support for finegrained timestamps.

gai_strerror

Purpose: Address and name information error description.

Synopsis:

```
#include <netdb.h>

const char *gai_strerror(int ecode);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

getc

Purpose: Get a byte from a stream.

Synopsis: `#include <stdio.h>`
`int getc(FILE *stream);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

getc_unlocked, getchar_unlocked, putc_unlocked, putchar_unlocked

Purpose: Stdio with explicit client locking.

cx Synopsis: `#include <stdio.h>`
`int getc_unlocked(FILE *stream);`
`int getchar_unlocked(void);`
`int putc_unlocked(int c, FILE *stream);`
`int putchar_unlocked(int c);`

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `getc_unlocked()`, `getchar_unlocked()`, `putc_unlocked()`, and `putchar_unlocked()` functions are moved from the Thread-Safe Functions option to the Base.

getchar

Purpose: Get a byte from a *stdin* stream.

Synopsis: `#include <stdio.h>`
`int getchar(void);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

getcwd

Purpose: Get the pathname of the current working directory.

Synopsis: `#include <unistd.h>`
`char *getcwd(char *buf, size_t size);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #140 is applied, changing the text for consistency with the *pwd* utility, adding text to address the case where the current directory is deeper in the file hierarchy than `{PATH_MAX}` bytes, and adding the requirements relating to pathnames beginning with two `<slash>` characters.

getdate

Purpose: Convert user format date and time.

XSI Synopsis:

```
#include <time.h>
struct tm *getdate(const char *string);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The description of the *getdate_err* value is expanded.

getdelim, getline

Purpose: Read a delimited record from *stream*.

CX Synopsis:

```
#include <stdio.h>
ssize_t getdelim(char **restrict lineptr, size_t *restrict n,
int delimiter, FILE *restrict stream);
ssize_t getline(char **restrict lineptr, size_t *restrict n,
FILE *restrict stream);
```

The *getdelim()* function reads from *stream* until it encounters a character matching the *delimiter* character.

The *getline()* function is equivalent to the *getdelim()* function with the *delimiter* character equal to the <newline> character.

These functions are widely used to solve the problem that the *fgets()* function has with long lines. The functions automatically enlarge the target buffers if needed. These are especially useful since they reduce code needed for applications.

Application writers should note that setting **lineptr* to a null pointer and **n* to zero are allowed and a recommended way to start parsing a file.

The *ferror()* or *feof()* functions should be used to distinguish between an error condition and an end-of-file condition.

Although a NUL terminator is always supplied after the line, note that *strlen(*lineptr)* will be smaller than the return value if the line contains embedded NUL characters.

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Issue 7: First released in Issue 7.

getegid

Purpose: Get the effective group ID.

Synopsis:

```
#include <unistd.h>
gid_t getegid(void);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

getenv

Purpose: Get value of an environment variable.

Synopsis: `#include <stdlib.h>`
`char *getenv(const char *name);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #062 is applied, clarifying that a call to `putenv()` may also cause the string to be overwritten.

Austin Group Interpretation 1003.1-2001 #148 is applied, adding the FUTURE DIRECTIONS.

geteuid

Purpose: Get the effective user ID.

Synopsis: `#include <unistd.h>`
`uid_t geteuid(void);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

getgid

Purpose: Get the real group ID.

Synopsis: `#include <unistd.h>`
`gid_t getgid(void);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

getgrgid, getgrgid_r

Purpose: Get group database entry for a group ID.

Synopsis: `#include <grp.h>`
`struct group *getgrgid(gid_t gid);`
`int getgrgid_r(gid_t gid, struct group *grp, char *buffer,`
`size_t bufsize, struct group **result);`

Derivation: First released in Issue 1. Derived from System V Release 2.0.

Issue 7: SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-166 is applied, changing `sysconf(_SC_GETGR_R_SIZE_MAX)` from the maximum size to an initial value suggested for the size, and adding an example of its use to the EXAMPLES section.

The `getgrgid_r()` function is moved from the Thread-Safe Functions option to the Base.

A minor addition is made to the EXAMPLES section, reminding the application developer to free memory allocated as if by `malloc()`.

getgrnam, getgrnam_r

Purpose: Search group database for a name.

Synopsis: `#include <grp.h>`

```
struct group *getgrnam(const char *name);
int getgrnam_r(const char *name, struct group *grp,
               char *buffer, size_t bufsize,
               struct group **result);
```

Derivation: First released in Issue 1. Derived from System V Release 2.0.

Issue 7: Austin Group Interpretation 1003.1-2001 #081 is applied, clarifying the RETURN VALUE section.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-166 is applied, changing `sysconf(_SC_GETGR_R_SIZE_MAX)` from the maximum size to an initial value suggested for the size, and adding an example of its use to the EXAMPLES section.

The `getgrnam_r()` function is moved from the Thread-Safe Functions option to the Base.

A minor addition is made to the EXAMPLES section, reminding the application developer to free memory allocated as if by `malloc()`.

getgroups

Purpose: Get supplementary group IDs.

Synopsis: `#include <unistd.h>`

```
int getgroups(int gidsetsize, gid_t grouplist[]);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

gethostid

Purpose: Get an identifier for the current host.

XS1 Synopsis: `#include <unistd.h>`
`long gethostid(void);`

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

gethostname

Purpose: Get name of current host.

Synopsis: `#include <unistd.h>`

```
int gethostname(char *name, size_t namelen);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

getitimer, setitimer

Purpose: Get and set value of interval timer.

OB XSI Synopsis:

```
#include <sys/time.h>

int getitimer(int which, struct itimerval *value);
int setitimer(int which,
               const struct itimerval *restrict value,
               struct itimerval *restrict ovalue);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The *getitimer()* and *setitimer()* functions are marked obsolescent. Applications should use the *timer_gettime()* and *timer_settime()* functions, respectively.

getlogin, getlogin_r

Purpose: Get login name.

Synopsis:

```
#include <unistd.h>

char *getlogin(void);
int getlogin_r(char *name, size_t namesize);
```

Derivation: First released in Issue 1. Derived from System V Release 2.0.

Issue 7: SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

The *getlogin_r()* function is moved from the Thread-Safe Functions option to the Base.

getmsg, getpmsg

Purpose: Receive next message from a STREAMS file (**STREAMS**).

OB XSR Synopsis:

```
#include <stropts.h>

int getmsg(int fildes, struct strbuf *restrict ctlptr,
           struct strbuf *restrict dataptr, int *restrict flagsp);
int getpmsg(int fildes, struct strbuf *restrict ctlptr,
            struct strbuf *restrict dataptr, int *restrict bandp,
            int *restrict flagsp);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The *getmsg()* and *getpmsg()* functions are marked obsolescent.

getnameinfo

Purpose: Get name information.

Synopsis:

```
#include <sys/socket.h>
#include <netdb.h>

int getnameinfo(const struct sockaddr *restrict sa,
                socklen_t salen, char *restrict node,
                socklen_t nodelen, char *restrict service,
```

```
socklen_t servicelen, int flags);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: SD5-XSH-ERN-127 is applied, clarifying the behavior if the address is the IPv6 unspecified address.

getopt, optarg, opterr, optind, optopt

Purpose: Command option parsing.

Synopsis: #include <unistd.h>

```
int getopt(int argc, char * const argv[], const char *optstring);
extern char *optarg;
extern int opterr, optind, optopt;
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

getpeername

Purpose: Get the name of the peer socket.

Synopsis: #include <sys/socket.h>

```
int getpeername(int socket, struct sockaddr *restrict address,
socklen_t *restrict address_len);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

getpgid

Purpose: Get the process group ID for a process.

Synopsis: #include <unistd.h>

```
pid_t getpgid(pid_t pid);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The *getpgid()* function is moved from the XSI option to the Base.

getpgrp

Purpose: Get the process group ID of the calling process.

Synopsis: #include <unistd.h>

```
pid_t getpgrp(void);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

getpid

Purpose: Get the process ID.

Synopsis: `#include <unistd.h>`
`pid_t getpid(void);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

getppid

Purpose: Get the parent process ID.

Synopsis: `#include <unistd.h>`
`pid_t getppid(void);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

getpriority, setpriority

Purpose: Get and set the nice value.

XSI Synopsis: `#include <sys/resource.h>`
`int getpriority(int which, id_t who);`
`int setpriority(int which, id_t who, int value);`

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

getpwnam, getpwnam_r

Purpose: Search user database for a name.

Synopsis: `#include <pwd.h>`
`struct passwd *getpwnam(const char *name);`
`int getpwnam_r(const char *name, struct passwd *pwd,`
`char *buffer, size_t bufsize,`
`struct passwd **result);`

Derivation: First released in Issue 1. Derived from System V Release 2.0.

Issue 7: SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-166 is applied, changing `sysconf(_SC_GETPW_R_SIZE_MAX)` from the maximum size to an initial value suggested for the size, and adding an example of its use to the EXAMPLES section.

The `getpwnam_r()` function is moved from the Thread-Safe Functions option to the Base.

A minor addition is made to the EXAMPLES section, reminding the application developer to free memory allocated as if by `malloc()`.

getpwuid, getpwuid_r

Purpose: Search user database for a user ID.

Synopsis: `#include <pwd.h>`

```
struct passwd *getpwuid(uid_t uid);
int getpwuid_r(uid_t uid, struct passwd *pwd, char *buffer,
               size_t bufsize, struct passwd **result);
```

Derivation: First released in Issue 1. Derived from System V Release 2.0.

Issue 7: SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-166 is applied, changing `sysconf(_SC_GETPW_R_SIZE_MAX)` from the maximum size to an initial value suggested for the size, and adding an example of its use to the EXAMPLES section.

The `getpwuid_r()` function is moved from the Thread-Safe Functions option to the Base.

A minor addition is made to the EXAMPLES section, reminding the application developer to free memory allocated as if by `malloc()`.

getrlimit, setrlimit

Purpose: Control maximum resource consumption.

XSI

Synopsis: `#include <sys/resource.h>`

```
int getrlimit(int resource, struct rlimit *rlp);
int setrlimit(int resource, const struct rlimit *rlp);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

getrusage

Purpose: Get information about resource utilization.

XSI

Synopsis: `#include <sys/resource.h>`

```
int getrusage(int who, struct rusage *r_usage);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

gets

Purpose: Get a string from a *stdin* stream.

OB

Synopsis: `#include <stdio.h>`

```
char *gets(char *s);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.

The *gets()* function is marked obsolescent. Applications should use the *fgets()* function instead.

Changes are made related to support for finegrained timestamps.

getsid

Purpose: Get the process group ID of a session leader.

Synopsis:

```
#include <unistd.h>

pid_t getsid(pid_t pid);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The *getsid()* function is moved from the XSI option to the Base.

getsockname

Purpose: Get the socket name.

Synopsis:

```
#include <sys/socket.h>

int getsockname(int socket, struct sockaddr *restrict address,
                socklen_t *restrict address_len);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

getsockopt

Purpose: Get the socket options.

Synopsis:

```
#include <sys/socket.h>

int getsockopt(int socket, int level,
               int option_name, void *restrict option_value,
               socklen_t *restrict option_len);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: Austin Group Interpretation 1003.1-2001 #158 is applied, removing text relating to socket options that is now in XSH Section 2.10.16 .

getsubopt

Purpose: Parse suboption arguments from a string.

Synopsis:

```
#include <stdlib.h>

int getsubopt(char **optionp, char * const *keylistp,
              char **valuep);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The *getsubopt()* function is moved from the XSI option to the Base.

gettimeofday

Purpose: Get the date and time.

OB XSI Synopsis:

```
#include <sys/time.h>

int gettimeofday(struct timeval *restrict tp,
                 void *restrict tzp);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The *gettimeofday()* function is marked obsolescent. Applications should use the *clock_gettime()* function instead.

getuid

Purpose: Get a real user ID.

Synopsis:

```
#include <unistd.h>

uid_t getuid(void);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

getwc

Purpose: Get a wide character from a stream.

Synopsis:

```
#include <stdio.h>
#include <wchar.h>

wint_t getwc(FILE *stream);
```

Derivation: First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working draft.

Issue 7: No functional changes are made in this issue.

getwchar

Purpose: Get a wide character from a *stdin* stream.

Synopsis:

```
#include <wchar.h>

wint_t getwchar(void);
```

Derivation: First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working draft.

Issue 7: No functional changes are made in this issue.

glob, globfree

Purpose: Generate pathnames matching a pattern.

Synopsis:

```
#include <glob.h>

int glob(const char *restrict pattern, int flags,
         int(*errfunc)(const char *epath, int errno),
         glob_t *restrict pglob);
void globfree(glob_t *pglob);
```

Derivation: First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 7: No functional changes are made in this issue.

gmtime, gmtime_r

Purpose: Convert a time value to a broken-down UTC time.

Synopsis: #include <time.h>

CX

```
struct tm *gmtime(const time_t *timer);
struct tm *gmtime_r(const time_t *restrict timer,
                    struct tm *restrict result);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *gmtime_r()* function is moved from the Thread-Safe Functions option to the Base.

grantpt

Purpose: Grant access to the slave pseudo-terminal device.

XSI

Synopsis: #include <stdlib.h>

```
int grantpt(int fildes);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

hcreate, hdestroy, hsearch

Purpose: Manage hash search table.

XSI

Synopsis: #include <search.h>

```
int hcreate(size_t nel);
void hdestroy(void);
ENTRY *hsearch(ENTRY item, ACTION action);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

htonl, htons, ntohl, ntohs

Purpose: Convert values between host and network byte order.

Synopsis: #include <arpa/inet.h>

```
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

hypot, hypotf, hypotl

Purpose: Euclidean distance function.

Synopsis:

```
#include <math.h>

double hypot(double x, double y);
float hypotf(float x, float y);
long double hypotl(long double x, long double y);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

iconv

Purpose: Codeset conversion function.

Synopsis:

```
#include <iconv.h>

size_t iconv(iconv_t cd, char **restrict inbuf,
             size_t *restrict inbytesleft, char **restrict outbuf,
             size_t *restrict outbytesleft);
```

Derivation: First released in Issue 4. Derived from the HP-UX Manual.

Issue 7: The *iconv()* function is moved from the XSI option to the Base.

iconv_close

Purpose: Codeset conversion deallocation function.

Synopsis:

```
#include <iconv.h>

int iconv_close(iconv_t cd);
```

Derivation: First released in Issue 4. Derived from the HP-UX Manual.

Issue 7: The *iconv_close()* function is moved from the XSI option to the Base.

iconv_open

Purpose: Codeset conversion allocation function.

Synopsis:

```
#include <iconv.h>

iconv_t iconv_open(const char *tocode, const char *fromcode);
```

Derivation: First released in Issue 4. Derived from the HP-UX Manual.

Issue 7: SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.
The *iconv_open()* function is moved from the XSI option to the Base.

if_freenameindex

Purpose: Free memory allocated by if_nameindex.

Synopsis: `#include <net/if.h>`
`void if_freenameindex(struct if_nameindex *ptr);`

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

if_indextoname

Purpose: Map a network interface index to its corresponding name.

Synopsis: `#include <net/if.h>`
`char *if_indextoname(unsigned ifindex, char *ifname);`

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

if_nameindex

Purpose: Return all network interface names and indexes.

Synopsis: `#include <net/if.h>`
`struct if_nameindex *if_nameindex(void);`

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

if_nametoindex

Purpose: Map a network interface name to its corresponding index.

Synopsis: `#include <net/if.h>`
`unsigned if_nametoindex(const char *ifname);`

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

ilogb, ilogbf, ilogbl

Purpose: Return an unbiased exponent.

Synopsis: `#include <math.h>`
`int ilogb(double x);`
`int ilogbf(float x);`
`int ilogbl(long double x);`

Derivation: First released in Issue 4, Version 2.

Issue 7: ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #48 (SD5-XSH-ERN-71), #49, and #79 (SD5-XSH-ERN-72) are applied.

imaxabs

Purpose: Return absolute value.

Synopsis: `#include <inttypes.h>`
`intmax_t imaxabs(intmax_t j);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

imaxdiv

Purpose: Return quotient and remainder.

Synopsis: `#include <inttypes.h>`
`imaxdiv_t imaxdiv(intmax_t numer, intmax_t denom);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

inet_addr, inet_ntoa

Purpose: IPv4 address manipulation.

Synopsis: `#include <arpa/inet.h>`
`in_addr_t inet_addr(const char *cp);`
`char *inet_ntoa(struct in_addr in);`

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

inet_ntop, inet_pton

Purpose: Convert IPv4 and IPv6 addresses between binary and text form.

Synopsis: `#include <arpa/inet.h>`
`const char *inet_ntop(int af, const void *restrict src,`
`char *restrict dst, socklen_t size);`
`int inet_pton(int af, const char *restrict src,`
`void *restrict dst);`

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

initstate, random, setstate, srandom

Purpose: Pseudo-random number functions.

XSI Synopsis: `#include <stdlib.h>`
`char *initstate(unsigned seed, char *state, size_t size);`
`long random(void);`
`char *setstate(char *state);`
`void srandom(unsigned seed);`

Derivation: First released in Issue 4, Version 2.

Issue 7: The type of the first argument to `setstate()` is changed from **const char *** to **char ***.

insque, remque

Purpose: Insert or remove an element in a queue.

XSI Synopsis:

```
#include <search.h>
void insque(void *element, void *pred);
void remque(void *element);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

ioctl

Purpose: Control a STREAMS device (**STREAMS**).

OB XSR Synopsis:

```
#include <stropts.h>
int ioctl(int fildes, int request, ... /* arg */);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #155 is applied, adding a “may fail” [EINVAL] error condition for the `I_SENDFD` command.

SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

The `ioctl()` function is marked obsolescent.

isalnum, isalnum_l

Purpose: Test for an alphanumeric character.

CX Synopsis:

```
#include <ctype.h>
int isalnum(int c);
int isalnum_l(int c, locale_t locale);
```

The `isalnum_l()` function tests whether `c` is a character of class **alpha** or **digit** in the locale represented by `locale`. A handle for use as `locale` can be obtained using `newlocale()` or `duplocale()`.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The `isalnum_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

isalpha, isalpha_l

Purpose: Test for an alphabetic character.

Synopsis: `#include <ctype.h>`

```
int isalpha(int c);
cx int isalpha_l(int c, locale_t locale);
```

The *isalpha_l()* function tests whether *c* is a character of class **alpha** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *isalpha_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

isascii

Purpose: Test for a 7-bit US-ASCII character.

```
OB XSI Synopsis: #include <ctype.h>
int isascii(int c);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *isascii()* function is marked obsolescent.

isastream

Purpose: Test a file descriptor (**STREAMS**).

```
OB XSR Synopsis: #include <stropts.h>
int isastream(int fildes);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The *isastream()* function is marked obsolescent.

isatty

Purpose: Test for a terminal device.

```
Synopsis: #include <unistd.h>
int isatty(int fildes);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

isblank, isblank_l

Purpose: Test for a blank character.

Synopsis: `#include <ctype.h>`

```
int isblank(int c);
cx int isblank_l(int c, locale_t locale);
```

The *isblank_l()* function tests whether *c* is a character of class **blank** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: The *isblank_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

iscntrl, iscntrl_l

Purpose: Test for a control character.

Synopsis: `#include <ctype.h>`

```
int iscntrl(int c);
cx int iscntrl_l(int c, locale_t locale);
```

The *iscntrl_l()* function tests whether *c* is a character of class **cntrl** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *iscntrl_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

isdigit, isdigit_l

Purpose: Test for a decimal digit.

Synopsis: `#include <ctype.h>`

```
int isdigit(int c);
cx int isdigit_l(int c, locale_t locale);
```

The *isdigit_l()* function tests whether *c* is a character of class **digit** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *isdigit_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

isfinite

Purpose: Test for finite value.

Synopsis: `#include <math.h>`
`int isfinite(real-floating x);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

isgraph, isgraph_l

Purpose: Test for a visible character.

Synopsis: `#include <ctype.h>`
`int isgraph(int c);`
`int isgraph_l(int c, locale_t locale);`

cx

The *isgraph_l()* function tests whether *c* is a character of class **graph** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *isgraph_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

isgreater

Purpose: Test if *x* greater than *y*.

Synopsis: `#include <math.h>`
`int isgreater(real-floating x, real-floating y);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

isgreaterequal

Purpose: Test if *x* is greater than or equal to *y*.

Synopsis: `#include <math.h>`
`int isgreaterequal(real-floating x, real-floating y);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

isinf

Purpose: Test for infinity.

Synopsis: `#include <math.h>`
`int isinf(real-floating x);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

isless

Purpose: Test if *x* is less than *y*.

Synopsis: `#include <math.h>`
`int isless(real-floating x, real-floating y);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

islessequal

Purpose: Test if *x* is less than or equal to *y*.

Synopsis: `#include <math.h>`
`int islessequal(real-floating x, real-floating y);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

islessgreater

Purpose: Test if *x* is less than or greater than *y*.

Synopsis: `#include <math.h>`
`int islessgreater(real-floating x, real-floating y);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

islower, islower_l

Purpose: Test for a lowercase letter.

Synopsis: `#include <ctype.h>`
`int islower(int c);`
`int islower_l(int c, locale_t locale);`

CX

The `islower_l()` function tests whether *c* is a character of class **lower** in the locale represented by *locale*. A handle for use as *locale* can be obtained using `newlocale()` or `duplocale()`.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *islower_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

isnan

Purpose: Test for a NaN.

Synopsis:

```
#include <math.h>
int isnan(real-floating x);
```

Derivation: First released in Issue 3.

Issue 7: No functional changes are made in this issue.

isnormal

Purpose: Test for a normal value.

Synopsis:

```
#include <math.h>
int isnormal(real-floating x);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

isprint, isprint_l

Purpose: Test for a printable character.

Synopsis:

```
#include <ctype.h>
int isprint(int c);
int isprint_l(int c, locale_t locale);
```

CX

The *isprint_l()* function tests whether *c* is a character of class **print** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *isprint_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

ispunct, ispunct_l

Purpose: Test for a punctuation character.

Synopsis:

```
#include <ctype.h>
int ispunct(int c);
int ispunct_l(int c, locale_t locale);
```

CX

The *ispunct_l()* function tests whether *c* is a character of class **punct** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *ispunct_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

isspace, isspace_l

Purpose: Test for a white-space character.

Synopsis: `#include <ctype.h>`

```

int isspace(int c);
CX int isspace_l(int c, locale_t locale);

```

The *isspace_l()* function tests whether *c* is a character of class **space** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *isspace_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

isunordered

Purpose: Test if arguments are unordered.

Synopsis: `#include <math.h>`

```
int isunordered(real-floating x, real-floating y);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

Issue 7: No functional changes are made in this issue.

isupper, isupper_l

Purpose: Test for an uppercase letter.

Synopsis: `#include <ctype.h>`

```

int isupper(int c);
CX int isupper_l(int c, locale_t locale);

```

The *isupper_l()* function tests whether *c* is a character of class **upper** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *isupper_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

iswalnum, iswalnum_l

Purpose: Test for an alphanumeric wide-character code.

Synopsis: `#include <wctype.h>`

```

CX      int iswalnum(wint_t wc);
        int iswalnum_l(wint_t wc, locale_t locale);

```

The *iswalnum_l()* function tests whether *wc* is a wide-character code representing a character of class **alpha** or **digit** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released as a World-wide Portability Interface in Issue 4.

Issue 7: The *iswalnum_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

iswalpha, iswalpha_l

Purpose: Test for an alphabetic wide-character code.

Synopsis: `#include <wctype.h>`

```

CX      int iswalpha(wint_t wc);
        int iswalpha_l(wint_t wc, locale_t locale);

```

The *iswalpha_l()* function tests whether *wc* is a wide-character code representing a character of class **alpha** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 4.

Issue 7: The *iswalpha_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

iswblank, iswblank_l

Purpose: Test for a blank wide-character code.

Synopsis: `#include <wctype.h>`

```

CX      int iswblank(wint_t wc);
        int iswblank_l(wint_t wc, locale_t locale);

```

The *iswblank_l()* function tests whether *wc* is a wide-character code representing a character of class **blank** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: The *iswblank_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

iswcntrl, iswcntrl_l

Purpose: Test for a control wide-character code.

Synopsis: #include <wctype.h>

```

CX      int iswcntrl(wint_t wc);
        int iswcntrl_l(wint_t wc, locale_t locale);

```

The *iswcntrl_l()* function tests whether *wc* is a wide-character code representing a character of class **cntrl** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 4.

Issue 7: The *iswcntrl_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

iswctype, iswctype_l

Purpose: Test character for a specified class.

Synopsis: #include <wctype.h>

```

CX      int iswctype(wint_t wc, wctype_t charclass);
        int iswctype_l(wint_t wc, wctype_t charclass,
                       locale_t locale);

```

The *iswctype_l()* function determines whether the wide-character code *wc* has the character class *charclass* in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released as World-wide Portability Interfaces in Issue 4.

Issue 7: The *iswctype_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

iswdigit, iswdigit_l

Purpose: Test for a decimal digit wide-character code.

Synopsis: #include <wctype.h>

```

CX      int iswdigit(wint_t wc);
        int iswdigit_l(wint_t wc, locale_t locale);

```

The *iswdigit_l()* function tests whether *wc* is a wide-character code representing a character of class **digit** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 4.

Issue 7: The *iswdigit_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

iswgraph, iswgraph_l

Purpose: Test for a visible wide-character code.

Synopsis: `#include <wctype.h>`

```

cx      int iswgraph(wint_t wc);
        int iswgraph_l(wint_t wc, locale_t locale);

```

The *iswgraph_l()* function tests whether *wc* is a wide-character code representing a character of class **graph** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 4.

Issue 7: The *iswgraph_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

iswlower, iswlower_l

Purpose: Test for a lowercase letter wide-character code.

Synopsis: `#include <wctype.h>`

```

cx      int iswlower(wint_t wc);
        int iswlower_l(wint_t wc, locale_t locale);

```

The *iswlower_l()* function tests whether *wc* is a wide-character code representing a character of class **lower** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 4.

Issue 7: The *iswlower_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

iswprint, iswprint_l

Purpose: Test for a printable wide-character code.

Synopsis: `#include <wctype.h>`

```

cx      int iswprint(wint_t wc);
        int iswprint_l(wint_t wc, locale_t locale);

```

The *iswprint_l()* function tests whether *wc* is a wide-character code representing a character of class **print** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 4.

Issue 7: The *iswprint_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

iswpunct, iswpunct_l

Purpose: Test for a punctuation wide-character code.

Synopsis: `#include <wctype.h>`

```

CX      int iswpunct(wint_t wc);
        int iswpunct_l(wint_t wc, locale_t locale);

```

The *iswpunct_l()* function tests whether *wc* is a wide-character code representing a character of class **punct** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 4.

Issue 7: The *iswpunct_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

iswspace, iswspace_l

Purpose: Test for a white-space wide-character code.

Synopsis: `#include <wctype.h>`

```

CX      int iswspace(wint_t wc);
        int iswspace_l(wint_t wc, locale_t locale);

```

The *iswspace_l()* function tests whether *wc* is a wide-character code representing a character of class **space** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 4.

Issue 7: The *iswspace_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

iswupper, iswupper_l

Purpose: Test for an uppercase letter wide-character code.

Synopsis: `#include <wctype.h>`

```

CX      int iswupper(wint_t wc);
        int iswupper_l(wint_t wc, locale_t locale);

```

The *iswupper_l()* function tests whether *wc* is a wide-character code representing a character of class **upper** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 4.

Issue 7: The *iswupper_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

iswxdigit, iswxdigit_l

Purpose: Test for a hexadecimal digit wide-character code.

Synopsis: `#include <wctype.h>`

CX `int iswxdigit(wint_t wc);`
`int iswxdigit_l(wint_t wc, locale_t locale);`

The *iswxdigit_l()* function tests whether *wc* is a wide-character code representing a character of class **xdigit** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 4.

Issue 7: The *iswxdigit_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

isxdigit, isxdigit_l

Purpose: Test for a hexadecimal digit.

Synopsis: `#include <ctype.h>`

CX `int isxdigit(int c);`
`int isxdigit_l(int c, locale_t locale);`

The *isxdigit_l()* function tests whether *c* is a character of class **xdigit** in the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *isxdigit_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

j0, j1, jn

Purpose: Bessel functions of the first kind.

XSI `Synopsis: #include <math.h>`
`double j0(double x);`
`double j1(double x);`
`double jn(int n, double x);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

kill

Purpose: Send a signal to a process or a group of processes.

cx Synopsis:

```
#include <signal.h>
int kill(pid_t pid, int sig);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

killpg

Purpose: Send a signal to a process group.

xsi Synopsis:

```
#include <signal.h>
int killpg(pid_t pgrp, int sig);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

labs, llabs

Purpose: Return a long integer absolute value.

Synopsis:

```
#include <stdlib.h>
long labs(long i);
long long llabs(long long i);
```

Derivation: First released in Issue 4. Derived from the ISO C standard.

Issue 7: No functional changes are made in this issue.

lchown

Purpose: Change the owner and group of a symbolic link.

Synopsis:

```
#include <unistd.h>
int lchown(const char *path, uid_t owner, gid_t group);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *lchown()* function is moved from the XSI option to the Base.

The [EOPNOTSUPP] error is removed.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

ldexp, ldexpf, ldexpl

Purpose: Load exponent of a floating-point number.

Synopsis: `#include <math.h>`

```
double ldexp(double x, int exp);
float ldexpf(float x, int exp);
long double ldexpl(long double x, int exp);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

ldiv, lldiv

Purpose: Compute quotient and remainder of a long division.

Synopsis: `#include <stdlib.h>`

```
ldiv_t ldiv(long numer, long denom);
lldiv_t lldiv(long long numer, long long denom);
```

Derivation: First released in Issue 4. Derived from the ISO C standard.

Issue 7: No functional changes are made in this issue.

lgamma, lgammaf, lgammal, signgam

Purpose: Log gamma function.

Synopsis: `#include <math.h>`

```
double lgamma(double x);
float lgammaf(float x);
long double lgammal(long double x);
extern int signgam;
```

XSI

Derivation: First released in Issue 3.

Issue 7: The DESCRIPTION is clarified regarding the value of *signgam* when *x* is Nan, $-\text{Inf}$, or a negative integer.

link, linkat

Purpose: Link one file to another file relative to two directory file descriptors.

Synopsis: `#include <unistd.h>`

```
int link(const char *path1, const char *path2);
int linkat(int fd1, const char *path1, int fd2,
           const char *path2, int flag);
```

The *linkat()* function is equivalent to the *link()* function except in the case where either *path1* or *path2* or both are relative paths. In this case a relative path *path1* is interpreted relative to the directory associated with the file descriptor *fd1* instead of the current working directory and similarly for *path2* and the file descriptor *fd2*. If the file descriptor was opened without `O_SEARCH`, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with `O_SEARCH`, the function does not perform the check.

The `AT_SYMLINK_FOLLOW` flag can be used to specify that if *path1* names a symbolic link, a new link for the target of the symbolic link is created. By default a new link for the symbolic link itself is created.

The purpose of the `linkat()` function is to link files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to `link()`, resulting in unspecified behavior. By opening a file descriptor for the directory of both the existing file and the target location and using the `linkat()` function it can be guaranteed that the both filenames are in the desired directories.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: If *path1* names a symbolic link, the `link()` function is no longer required to follow the link: it is implementation-defined whether `link()` follows the link, or creates a new link to the symbolic link itself. Applications which need control over whether the link is followed can use the new `linkat()` function, setting the *flag* argument appropriately.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than `{PATH_MAX}`.

The `linkat()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Functionality relating to XSI STREAMS is marked obsolescent.

Changes are made related to support for finegrained timestamps.

lio_listio

Purpose: List directed I/O.

Synopsis: `#include <aio.h>`

```
int lio_listio(int mode,
              struct aiocb *restrict const list[restrict],
              int nent, struct sigevent *restrict sig);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: The `lio_listio()` function is moved from the Asynchronous Input and Output option to the Base.

listen

Purpose: Listen for socket connections and limit the queue of incoming connections.

Synopsis: `#include <sys/socket.h>`

```
int listen(int socket, int backlog);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

llrint, llrintf, llrintl

Purpose: Round to the nearest integer value using current rounding direction.

Synopsis: `#include <math.h>`
`long long llrint(double x);`
`long long llrintf(float x);`
`long long llrintl(long double x);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #53 is applied.

llround, llroundf, llroundl

Purpose: Round to nearest integer value.

Synopsis: `#include <math.h>`
`long long llround(double x);`
`long long llroundf(float x);`
`long long llroundl(long double x);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #54 (SD5-XSH-ERN-75) is applied.

localeconv

Purpose: Return locale-specific information.

Synopsis: `#include <locale.h>`
`struct lconv *localeconv(void);`

Derivation: First released in Issue 4. Derived from the ANSI C standard.

Issue 7: The definitions of **int_curr_symbol** and **currency_symbol** are updated.
 The examples in the APPLICATION USAGE section are updated.

localtime, localtime_r

Purpose: Convert a time value to a broken-down local time.

Synopsis: `#include <time.h>`
`struct tm *localtime(const time_t *timer);`
 CX `struct tm *localtime_r(const time_t *restrict timer,`
`struct tm *restrict result);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The `localtime_r()` function is moved from the Thread-Safe Functions option to the Base.
 Changes are made to the EXAMPLES section related to support for finegrained timestamps.

lockf

Purpose: Record locking on files.

XS1 Synopsis:

```
#include <unistd.h>
int lockf(int fildev, int function, off_t size);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #054 is applied, updating the DESCRIPTION to change “other threads” to “threads in other processes”.

log, logf, logl

Purpose: Natural logarithm function.

Synopsis:

```
#include <math.h>
double log(double x);
float logf(float x);
long double logl(long double x);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

log10, log10f, log10l

Purpose: Base 10 logarithm function.

Synopsis:

```
#include <math.h>
double log10(double x);
float log10f(float x);
long double log10l(long double x);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

log1p, log1pf, log1pl

Purpose: Compute a natural logarithm.

Synopsis:

```
#include <math.h>
double log1p(double x);
float log1pf(float x);
long double log1pl(long double x);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

log2, log2f, log2l

Purpose: Compute base 2 logarithm functions.

Synopsis: `#include <math.h>`
`double log2(double x);`
`float log2f(float x);`
`long double log2l(long double x);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

logb, logbf, logbl

Purpose: Radix-independent exponent.

Synopsis: `#include <math.h>`
`double logb(double x);`
`float logbf(float x);`
`long double logbl(long double x);`

Derivation: First released in Issue 4, Version 2.

Issue 7: ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #50 (SD5-XSH-ERN-76) is applied.

longjmp

Purpose: Non-local goto.

Synopsis: `#include <setjmp.h>`
`void longjmp(jmp_buf env, int val);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

lrint, lrintf, lrintl

Purpose: Round to nearest integer value using current rounding direction.

Synopsis: `#include <math.h>`
`long lrint(double x);`
`long lrintf(float x);`
`long lrintl(long double x);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #53 (SD5-XSH-ERN-77) is applied.

lround, lroundf, lroundl

Purpose: Round to nearest integer value.

Synopsis: `#include <math.h>`
`long lround(double x);`
`long lroundf(float x);`
`long lroundl(long double x);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #54 (SD5-XSH-ERN-78) is applied.

lsearch, lfind

Purpose: Linear search and update.

XSI Synopsis: `#include <search.h>`
`void *lsearch(const void *key, void *base, size_t *nelp,`
`size_t width, int (*compar)(const void *, const void *));`
`void *lfind(const void *key, const void *base, size_t *nelp,`
`size_t width, int (*compar)(const void *, const void *));`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

lseek

Purpose: Move the read/write file offset.

Synopsis: `#include <unistd.h>`
`off_t lseek(int fildes, off_t offset, int whence);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

malloc

Purpose: A memory allocator.

Synopsis: `#include <stdlib.h>`
`void *malloc(size_t size);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

mblen

Purpose: Get number of bytes in a character.

Synopsis: `#include <stdlib.h>`
`int mblen(const char *s, size_t n);`

Derivation: First released in Issue 4. Aligned with the ISO C standard.

Issue 7: No functional changes are made in this issue.

mbrlen

Purpose: Get number of bytes in a character (restartable).

Synopsis: `#include <wchar.h>`
`size_t mbrlen(const char *restrict s, size_t n,
 mbstate_t *restrict ps);`

Derivation: First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a “may fail” to a “shall fail”.

mbrtowc

Purpose: Convert a character to a wide-character code (restartable).

Synopsis: `#include <wchar.h>`
`size_t mbrtowc(wchar_t *restrict pwc, const char *restrict s,
 size_t n, mbstate_t *restrict ps);`

Derivation: First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a “may fail” to a “shall fail”.

mbsinit

Purpose: Determine conversion object status.

Synopsis: `#include <wchar.h>`
`int mbsinit(const mbstate_t *ps);`

Derivation: First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: No functional changes are made in this issue.

mbsnrtowcs, mbsrtowcs

Purpose: Convert a character string to a wide-character string (restartable).

Synopsis: `#include <wchar.h>`

CX

```
size_t mbsnrtowcs(wchar_t *restrict dst,
                 const char **restrict src, size_t nmc,
                 size_t len, mbstate_t *restrict ps);
size_t mbsrtowcs(wchar_t *restrict dst,
                 const char **restrict src, size_t len,
                 mbstate_t *restrict ps);
```

The *mbsnrtowcs()* function is equivalent to the *mbsrtowcs()* function, except that the conversion of characters pointed to by *src* is limited to at most *nmc* bytes (the size of the input buffer).

Derivation: First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a “may fail” to a “shall fail”.

The *mbsnrtowcs()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

mbstowcs

Purpose: Convert a character string to a wide-character string.

Synopsis: `#include <stdlib.h>`

```
size_t mbstowcs(wchar_t *restrict pwcs, const char *restrict s,
                size_t n);
```

Derivation: First released in Issue 4. Aligned with the ISO C standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a “may fail” to a “shall fail”.

mbtowc

Purpose: Convert a character to a wide-character code.

Synopsis: `#include <stdlib.h>`

```
int mbtowc(wchar_t *restrict pwc, const char *restrict s,
            size_t n);
```

Derivation: First released in Issue 4. Aligned with the ISO C standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a “may fail” to a “shall fail”.

memcpy

Purpose: Copy bytes in memory.

XSI Synopsis:

```
#include <string.h>
void *memcpy(void *restrict s1, const void *restrict s2,
             int c, size_t n);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

memchr

Purpose: Find byte in memory.

Synopsis:

```
#include <string.h>
void *memchr(const void *s, int c, size_t n);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

memcmp

Purpose: Compare bytes in memory.

Synopsis:

```
#include <string.h>
int memcmp(const void *s1, const void *s2, size_t n);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

memcpy

Purpose: Copy bytes in memory.

Synopsis:

```
#include <string.h>
void *memcpy(void *restrict s1, const void *restrict s2,
             size_t n);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

memmove

Purpose: Copy bytes in memory with overlapping areas.

Synopsis:

```
#include <string.h>
void *memmove(void *s1, const void *s2, size_t n);
```

Derivation: First released in Issue 4. Derived from the ANSI C standard.

Issue 7: No functional changes are made in this issue.

memset

Purpose: Set bytes in memory.

Synopsis: `#include <string.h>`
`void *memset(void *s, int c, size_t n);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

mkdir, mkdirat

Purpose: Make a directory relative to directory file descriptor.

Synopsis: `#include <sys/stat.h>`
`int mkdir(const char *path, mode_t mode);`
`int mkdirat(int fd, const char *path, mode_t mode);`

The `mkdirat()` function is equivalent to the `mkdir()` function except in the case where `path` specifies a relative path. In this case the newly created directory is created relative to the directory associated with the file descriptor `fd` instead of the current working directory. If the file descriptor was opened without `O_SEARCH`, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with `O_SEARCH`, the function does not perform the check.

The purpose of the `mkdirat()` function is to create a directory in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to the call to `mkdir()`, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the `mkdirat()` function it can be guaranteed that the newly created directory is located relative to the desired directory.

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than `{PATH_MAX}`.

The `mkdirat()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

mkdtemp, mkstemp

Purpose: Create a unique directory or file.

cx Synopsis: `#include <stdlib.h>`
`char *mkdtemp(char *template);`
`int mkstemp(char *template);`

The `mkdtemp()` function uses the contents of `template` to construct a unique directory name. The string provided in `template` is a filename ending with six trailing 'x's. The `mkdtemp()` function replaces each 'x' with a character from the portable filename character set. The characters are chosen such that the resulting name does not duplicate the name of an existing file at the time of a call to `mkdtemp()`. The unique directory name is used to attempt to create the

directory using mode 0700 as modified by the file creation mask.

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

SD5-XSH-ERN-168 is applied, clarifying file permissions upon creation.

The *mkstemp()* function is moved from the XSI option to the Base.

The *mkdtemp()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

mkfifo, mkfifoat

Purpose: Make a FIFO special file relative to directory file descriptor.

Synopsis: `#include <sys/stat.h>`

```
int mkfifo(const char *path, mode_t mode);
int mkfifoat(int fd, const char *path, mode_t mode);
```

The *mkfifoat()* function is equivalent to the *mkfifo()* function except in the case where *path* specifies a relative path. In this case the newly created FIFO is created relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was opened without O_SEARCH, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O_SEARCH, the function does not perform the check.

The purpose of the *mkfifoat()* function is to create a FIFO special file in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *mkfifo()*, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *mkfifoat()* function it can be guaranteed that the newly created FIFO is located relative to the desired directory.

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *mkfifoat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

mknod, mknodat

Purpose: Make directory, special file, or regular file.

XSI

Synopsis: `#include <sys/stat.h>`

```
int mknod(const char *path, mode_t mode, dev_t dev);
int mknodat(int fd, const char *path, mode_t mode, dev_t dev);
```

The *mknodat()* function is equivalent to the *mknod()* function except in the case where *path* specifies a relative path. In this case the newly created directory, special file, or regular file is located relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the file descriptor was

opened without `O_SEARCH`, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with `O_SEARCH`, the function does not perform the check.

The purpose of the `mknodat()` function is to create directories, special files, or regular files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to `mknod()`, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the `mknodat()` function it can be guaranteed that the newly created directory, special file, or regular file is located relative to the desired directory.

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than `{PATH_MAX}`.

The `mknodat()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

mktime

Purpose: Convert broken-down time into time since the Epoch.

Synopsis: `#include <time.h>`

```
time_t mktime(struct tm *timeptr);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ANSI C standard.

Issue 7: No functional changes are made in this issue.

mlock, munlock

Purpose: Lock or unlock a range of process address space (**REALTIME**).

MLR

Synopsis: `#include <sys/mman.h>`

```
int mlock(const void *addr, size_t len);
int munlock(const void *addr, size_t len);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: No functional changes are made in this issue.

mlockall, munlockall

Purpose: Lock/unlock the address space of a process (**REALTIME**).

ML

Synopsis: `#include <sys/mman.h>`

```
int mlockall(int flags);
int munlockall(void);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: No functional changes are made in this issue.

mmap

Purpose: Map pages of memory.

Synopsis:

```
#include <sys/mman.h>

void *mmap(void *addr, size_t len, int prot, int flags,
           int fildes, off_t off);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretations 1003.1-2001 #078 and #079 are applied, clarifying page alignment requirements and adding a note about the state of synchronization objects becoming undefined when a shared region is unmapped.

Functionality relating to the Memory Protection and Memory Mapped Files options is moved to the Base.

Changes are made related to support for finegrained timestamps.

modf, modff, modfl

Purpose: Decompose a floating-point number.

Synopsis:

```
#include <math.h>

double modf(double x, double *iptr);
float modff(float value, float *iptr);
long double modfl(long double value, long double *iptr);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

mprotect

Purpose: Set protection of memory mapping.

Synopsis:

```
#include <sys/mman.h>

int mprotect(void *addr, size_t len, int prot);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment requirements.

The *mprotect()* function is moved from the Memory Protection option to the Base.

mq_close

Purpose: Close a message queue (**REALTIME**).

MSG Synopsis:

```
#include <mqqueue.h>
int mq_close(mqd_t mqdes);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: No functional changes are made in this issue.

mq_getattr

Purpose: Get message queue attributes (**REALTIME**).

MSG Synopsis:

```
#include <mqqueue.h>
int mq_getattr(mqd_t mqdes, struct mq_attr *mqstat);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: No functional changes are made in this issue.

mq_notify

Purpose: Notify process that a message is available (**REALTIME**).

MSG Synopsis:

```
#include <mqqueue.h>
int mq_notify(mqd_t mqdes,
const struct sigevent *notification);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: SD5-XSH-ERN-38 is applied, adding the *mq_timedreceive()* function to the DESCRIPTION.

Austin Group Interpretation 1003.1-2001 #032 is applied, adding the [EINVAL] error.

An example is added.

mq_open

Purpose: Open a message queue (**REALTIME**).

MSG Synopsis:

```
#include <mqqueue.h>
mqd_t mq_open(const char *name, int oflag, ...);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error.

Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the user ID and group ID of the message queue.

mq_receive, mq_timedreceive

Purpose: Receive a message from a message queue (**REALTIME**).

MSG Synopsis:

```
#include <mqqueue.h>

ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
    unsigned *msg_prio);

#include <mqqueue.h>
#include <time.h>

ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
    size_t msg_len, unsigned *restrict msg_prio,
    const struct timespec *restrict abstime);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: The *mq_timedreceive()* function is moved from the Timeouts option to the Base. Functionality relating to the Timers option is moved to the Base.

mq_send, mq_timedsend

Purpose: Send a message to a message queue (**REALTIME**).

MSG Synopsis:

```
#include <mqqueue.h>

int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
    unsigned msg_prio);

#include <mqqueue.h>
#include <time.h>

int mq_timedsend(mqd_t mqdes, const char *msg_ptr,
    size_t msg_len, unsigned msg_prio,
    const struct timespec *abstime);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: The *mq_timedsend()* function is moved from the Timeouts option to the Base. Functionality relating to the Timers option is moved to the Base.

mq_setattr

Purpose: Set message queue attributes (**REALTIME**).

MSG Synopsis:

```
#include <mqqueue.h>
int mq_setattr(mqd_t mqdes,
               const struct mq_attr *restrict mqstat,
               struct mq_attr *restrict omqstat);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: No functional changes are made in this issue.

mq_unlink

Purpose: Remove a message queue (**REALTIME**).

MSG Synopsis:

```
#include <mqqueue.h>
int mq_unlink(const char *name);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error .

Austin Group Interpretation 1003.1-2001 #141 is applied, requiring that after a successful call to *mq_unlink()*, reuse of the name shall subsequently cause *mq_open()* to behave as if no message queue of that name exists (that is, *mq_open()* will fail if *O_CREAT* is not set, or will create a new message queue if *O_CREAT* is set). Previously, attempts to recreate the message queue were allowed to fail.

msgctl

Purpose: XSI message control operations.

XSI Synopsis:

```
#include <sys/msg.h>
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

Derivation: First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 7: No functional changes are made in this issue.

msgget

Purpose: Get the XSI message queue identifier.

XSI Synopsis:

```
#include <sys/msg.h>
int msgget(key_t key, int msgflg);
```

Derivation: First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 7: No functional changes are made in this issue.

msgrcv

Purpose: XSI message receive operation.

XSI Synopsis:

```
#include <sys/msg.h>
ssize_t msgrcv(int msqid, void *msgp, size_t msgsz,
               long msgtyp, int msgflg);
```

Derivation: First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 7: No functional changes are made in this issue.

msgsnd

Purpose: XSI message send operation.

XSI Synopsis:

```
#include <sys/msg.h>
int msgsnd(int msqid, const void *msgp, size_t msgsz,
            int msgflg);
```

Derivation: First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 7: No functional changes are made in this issue.

msync

Purpose: Synchronize memory with physical storage.

XSI|SIO Synopsis:

```
#include <sys/mman.h>
int msync(void *addr, size_t len, int flags);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment requirements.

The *msync()* function is marked as part of the Synchronized Input and Output option or XSI option as the Memory Mapped Files is moved to the Base.

Changes are made related to support for finegrained timestamps.

munmap

Purpose: Unmap pages of memory.

Synopsis:

```
#include <sys/mman.h>
int munmap(void *addr, size_t len);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment requirements.

The *munmap()* function is moved from the Memory Mapped Files option to the Base.

nan, nanf, nanl

Purpose: Return quiet NaN.

Synopsis: `#include <math.h>`
`double nan(const char *tagp);`
`float nanf(const char *tagp);`
`long double nanl(const char *tagp);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

nanosleep

Purpose: High resolution sleep.

cx Synopsis: `#include <time.h>`
`int nanosleep(const struct timespec *rqtp,`
`struct timespec *rmtp);`

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: SD5-XBD-ERN-33 is applied, clarifying that the *rqtp* and *rmtp* arguments may point to the same object.

The *nanosleep()* function is moved from the Timers option to the Base.

nearbyint, nearbyintf, nearbyintl

Purpose: Floating-point rounding functions.

Synopsis: `#include <math.h>`
`double nearbyint(double x);`
`float nearbyintf(float x);`
`long double nearbyintl(long double x);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

newlocale

Purpose: Create or modify a locale object.

cx Synopsis: `#include <locale.h>`
`locale_t newlocale(int category_mask, const char *locale,`
`locale_t base);`

The *newlocale()* function creates a new locale object or modifies an existing one.

Application writers should note that handles for locale objects created by the *newlocale()* function should be released by a corresponding call to *freelocale()*. Also, the special locale object *LC_GLOBAL_LOCALE* must not be passed for the *base* argument, even when returned by the *uselocale()* function.

The following example shows the construction of a locale where the *LC_CTYPE*

category data comes from a locale *loc1*, and the *LC_TIME* category data from a locale *tok2*:

```
#include <locale.h>
...
locale_t loc, new_loc;
/* Get the "loc1" data. */
loc = newlocale (LC_CTYPE_MASK, "loc1", NULL);
if (loc == (locale_t) 0)
    abort ();
/* Get the "loc2" data. */
new_loc = newlocale (LC_TIME_MASK, "loc2", loc);
if (new_loc != (locale_t) 0)
    /* We don't abort if this fails. In this case this
       simply used to unchanged locale object. */
    loc = new_loc;
...
```

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 4.

Issue 7: First released in Issue 7.

nextafter, nextafterf, nextafterl, nexttoward, nexttowardf, nexttowardl

Purpose: Next representable floating-point number.

Synopsis: #include <math.h>

```
double nextafter(double x, double y);
float nextafterf(float x, float y);
long double nextafterl(long double x, long double y);
double nexttoward(double x, long double y);
float nexttowardf(float x, long double y);
long double nexttowardl(long double x, long double y);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

nftw

Purpose: Walk a file tree.

XSI

Synopsis: #include <ftw.h>

```
int nftw(const char *path, int (*fn)(const char *,
    const struct stat *, int, struct FTW *), int fd_limit,
    int flags);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

APPLICATION USAGE is added and the EXAMPLES section is replaced with a new example.

nice

Purpose: Change the nice value of a process.

XSI Synopsis:

```
#include <unistd.h>
int nice(int incr);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

nl_langinfo, nl_langinfo_l

Purpose: Language information.

Synopsis:

```
#include <langinfo.h>
char *nl_langinfo(nl_item item);
char *nl_langinfo_l(nl_item item, locale_t locale);
```

Derivation: First released in Issue 2.

Issue 7: The `nl_langinfo()` function is moved from the XSI option to the Base.

The `nl_langinfo_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

open, openat

Purpose: Open file relative to directory file descriptor.

OH Synopsis:

```
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *path, int oflag, ...);
int openat(int fd, const char *path, int oflag, ...);
```

The `openat()` function is equivalent to the `open()` function except in the case where `path` specifies a relative path. In this case the file to be opened is determined relative to the directory associated with the file descriptor `fd` instead of the current working directory. If the file descriptor was opened without `O_SEARCH`, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with `O_SEARCH`, the function does not perform the check.

The purpose of the `openat()` function is to enable opening files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to `open()`, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the `openat()` function it can be guaranteed that the opened file is located relative to the desired directory.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #113 is applied, requiring the `O_SYNC` flag to be supported for regular files, even if the Synchronized Input and Output option is not supported.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

Austin Group Interpretation 1003.1-2001 #144 is applied, adding the O_TTY_INIT flag.

Austin Group Interpretation 1003.1-2001 #171 is applied, adding support to set the FD_CLOEXEC flag atomically at *open()*, and adding the F_DUPFD_CLOEXEC flag.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

This page is revised and the *openat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Functionality relating to the XSI STREAMS option is marked obsolescent.

Changes are made related to support for finegrained timestamps.

open_memstream, open_wmemstream

Purpose: Open a dynamic memory buffer stream.

cx

Synopsis:

```
#include <stdio.h>
FILE *open_memstream(char **bufp, size_t *sizep);
#include <wchar.h>
FILE *open_wmemstream(wchar_t **bufp, size_t *sizep);
```

The *open_memstream()* and *open_wmemstream()* functions create an I/O stream associated with a dynamically allocated memory buffer.

These functions are similar to *fmemopen()*, except that the memory is always allocated dynamically by the function, and the stream is opened only for output.

Application writers should note that the buffer created by these functions should be freed by the application after closing the stream, by means of a call to *free()*.

An example program using the *open_memstream()* interface follows:

```
#include <stdio.h>
#include <stdlib.h>

int
main (void)
{
    FILE *stream;
    char *buf;
    size_t len;
    off_t eob;

    stream = open_memstream (&buf, &len);
    if (stream == NULL)
        /* handle error */ ;
    fprintf (stream, "hello my world");
    fflush (stream);
    printf ("buf=%s, len=%zu\n", buf, len);
    eob = ftello(stream);
    fseeko (stream, 0, SEEK_SET);
```

```

        fprintf (stream, "good-bye");
        fseeko (stream, eob, SEEK_SET);
        fclose (stream);
        printf ("buf=%s, len=%zu\n", buf, len);
        free (buf);
        return 0;
    }

```

This program produces the following output:

```

buf=hello my world, len=14
buf=good-bye world, len=14

```

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Issue 7: First released in Issue 7.

pause

Purpose: Suspend the thread until a signal is received.

Synopsis: `#include <unistd.h>`
`int pause(void);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

pclose

Purpose: Close a pipe stream to or from a process.

cx Synopsis: `#include <stdio.h>`
`int pclose(FILE *stream);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

perror

Purpose: Write error messages to standard error.

Synopsis: `#include <stdio.h>`
`void perror(const char *s);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Changes are made related to support for finegrained timestamps.

pipe

Purpose: Create an interprocess channel.

Synopsis: `#include <unistd.h>`
`int pipe(int fildes[2]);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: SD5-XSH-ERN-156 is applied, updating the DESCRIPTION to state the setting of the pipe's user ID and group ID.

Changes are made related to support for finegrained timestamps.

poll

Purpose: Input/output multiplexing.

Synopsis: `#include <poll.h>`
`int poll(struct pollfd fds[], nfds_t nfds, int timeout);`

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #209 is applied, clarifying the POLLHUP event.

The `poll()` function is moved from the XSI option to the Base.

Functionality relating to the XSI STREAMS option is marked obsolescent.

popen

Purpose: Initiate pipe streams to or from a process.

cx Synopsis: `#include <stdio.h>`
`FILE *popen(const char *command, const char *mode);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #029 is applied, clarifying the values for `mode` in the DESCRIPTION.

SD5-XSH-ERN-149 is applied, changing the {STREAM_MAX} [EMFILE] error condition from a "may fail" to a "shall fail".

posix_fadvise

Purpose: File advisory information (**ADVANCED REALTIME**).

ADV Synopsis: `#include <fcntl.h>`
`int posix_fadvise(int fd, off_t offset, off_t len,`
`int advice);`

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: Austin Group Interpretation 1003.1-2001 #024 is applied, changing the definition of the [EINVAL] error.

posix_fallocate

Purpose: File space control (**ADVANCED REALTIME**).

ADV

Synopsis:

```
#include <fcntl.h>
int posix_fallocate(int fd, off_t offset, off_t len);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: Austin Group Interpretations 1003.1-2001 #022, #024, and #162 are applied, changing the definition of the [EINVAL] error.

posix_madvise

Purpose: Memory advisory information and alignment control (**ADVANCED REALTIME**).

ADV

Synopsis:

```
#include <sys/mman.h>
int posix_madvise(void *addr, size_t len, int advice);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

posix_mem_offset

Purpose: Find offset and length of a mapped typed memory block (**ADVANCED REALTIME**).

TYM

Synopsis:

```
#include <sys/mman.h>
int posix_mem_offset(const void *restrict addr, size_t len,
off_t *restrict off, size_t *restrict contig_len,
int *restrict fildes);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: No functional changes are made in this issue.

posix_memalign

Purpose: Aligned memory allocation (**ADVANCED REALTIME**).

ADV

Synopsis:

```
#include <stdlib.h>
int posix_memalign(void **memptr, size_t alignment,
size_t size);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: Austin Group Interpretation 1003.1-2001 #058 is applied, clarifying the value of the *alignment* argument in the DESCRIPTION.

Austin Group Interpretation 1003.1-2001 #152 is applied, clarifying the behavior when the size of the space requested is 0.

posix_openpt

Purpose: Open a pseudo-terminal device.

XSI Synopsis:

```
#include <stdlib.h>
#include <fcntl.h>

int posix_openpt(int oflag);
```

Derivation: First released in Issue 6.

Issue 7: SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.
SD5-XSH-ERN-51 is applied, correcting an error in the EXAMPLES section.

posix_spawn, posix_spawnp

Purpose: Spawn a process (**ADVANCED REALTIME**).

SPN Synopsis:

```
#include <spawn.h>

int posix_spawn(pid_t *restrict pid,
               const char *restrict path,
               const posix_spawn_file_actions_t *file_actions,
               const posix_spawnattr_t *restrict attrp,
               char *const argv[restrict], char *const envp[restrict]);
int posix_spawnp(pid_t *restrict pid,
                const char *restrict file,
                const posix_spawn_file_actions_t *file_actions,
                const posix_spawnattr_t *restrict attrp,
                char *const argv[restrict], char *const envp[restrict]);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: Functionality relating to the Threads option is moved to the Base.

posix_spawn_file_actions_addclose, posix_spawn_file_actions_addopen

Purpose: Add close or open action to spawn file actions object (**ADVANCED REALTIME**).

SPN Synopsis:

```
#include <spawn.h>

int posix_spawn_file_actions_addclose(
    posix_spawn_file_actions_t
    *file_actions, int fildes);
int posix_spawn_file_actions_addopen(
    posix_spawn_file_actions_t
    *restrict file_actions, int fildes,
    const char *restrict path, int oflag, mode_t mode);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

posix_spawn_file_actions_adddup2

Purpose: Add dup2 action to spawn file actions object (**ADVANCED REALTIME**).

SPN

Synopsis:

```
#include <spawn.h>

int posix_spawn_file_actions_adddup2(
    posix_spawn_file_actions_t
        *file_actions, int fildes, int newfildes);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

posix_spawn_file_actions_destroy, posix_spawn_file_actions_init

Purpose: Destroy and initialize spawn file actions object (**ADVANCED REALTIME**).

SPN

Synopsis:

```
#include <spawn.h>

int posix_spawn_file_actions_destroy(
    posix_spawn_file_actions_t
        *file_actions);
int posix_spawn_file_actions_init(posix_spawn_file_actions_t
    *file_actions);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

posix_spawnattr_destroy, posix_spawnattr_init

Purpose: Destroy and initialize spawn attributes object (**ADVANCED REALTIME**).

SPN

Synopsis:

```
#include <spawn.h>

int posix_spawnattr_destroy(posix_spawnattr_t *attr);
int posix_spawnattr_init(posix_spawnattr_t *attr);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

posix_spawnattr_getflags, posix_spawnattr_setflags

Purpose: Get and set the spawn-flags attribute of a spawn attributes object (**ADVANCED REALTIME**).

SPN

Synopsis:

```
#include <spawn.h>

int posix_spawnattr_getflags(
    const posix_spawnattr_t *restrict attr,
    short *restrict flags);
int posix_spawnattr_setflags(
    posix_spawnattr_t *attr, short flags);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

posix_spawnattr_getpgroup, posix_spawnattr_setpgroup

Purpose: Get and set the spawn-pgroup attribute of a spawn attributes object (**ADVANCED REALTIME**).

SPN

Synopsis:

```
#include <spawn.h>

int posix_spawnattr_getpgroup(
    const posix_spawnattr_t *restrict attr,
    pid_t *restrict pgroup);
int posix_spawnattr_setpgroup(posix_spawnattr_t *attr,
    pid_t pgroup);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

posix_spawnattr_getschedparam, posix_spawnattr_setschedparam

Purpose: Get and set the spawn-schedparam attribute of a spawn attributes object (**ADVANCED REALTIME**).

SPN PS

Synopsis:

```
#include <spawn.h>
#include <sched.h>

int posix_spawnattr_getschedparam(const posix_spawnattr_t
    *restrict attr, struct sched_param *restrict schedparam);
int posix_spawnattr_setschedparam(
    posix_spawnattr_t *restrict attr,
    const struct sched_param *restrict schedparam);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

posix_spawnattr_getschedpolicy, posix_spawnattr_setschedpolicy

Purpose: Get and set the spawn-schedpolicy attribute of a spawn attributes object (**ADVANCED REALTIME**).

SPN PS

Synopsis:

```
#include <spawn.h>
#include <sched.h>

int posix_spawnattr_getschedpolicy(const posix_spawnattr_t
    *restrict attr, int *restrict schedpolicy);
int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
    int schedpolicy);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

posix_spawnattr_getsigdefault, posix_spawnattr_setsigdefault

Purpose: Get and set the spawn-sigdefault attribute of a spawn attributes object (**ADVANCED REALTIME**).

SPN Synopsis:

```
#include <signal.h>
#include <spawn.h>

int posix_spawnattr_getsigdefault(const posix_spawnattr_t
    *restrict attr, sigset_t *restrict sigdefault);
int posix_spawnattr_setsigdefault(
    posix_spawnattr_t *restrict attr,
    const sigset_t *restrict sigdefault);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

posix_spawnattr_getsigmask, posix_spawnattr_setsigmask

Purpose: Get and set the spawn-sigmask attribute of a spawn attributes object (**ADVANCED REALTIME**).

SPN Synopsis:

```
#include <signal.h>
#include <spawn.h>

int posix_spawnattr_getsigmask(
    const posix_spawnattr_t *restrict attr,
    sigset_t *restrict sigmask);
int posix_spawnattr_setsigmask(
    posix_spawnattr_t *restrict attr,
    const sigset_t *restrict sigmask);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: No functional changes are made in this issue.

posix_trace_attr_destroy, posix_trace_attr_init

Purpose: Destroy and initialize the trace stream attributes object (**TRACING**).

OB TRC Synopsis:

```
#include <trace.h>

int posix_trace_attr_destroy(trace_attr_t *attr);
int posix_trace_attr_init(trace_attr_t *attr);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The *posix_trace_attr_destroy()* and *posix_trace_attr_init()* functions are marked obsolescent.

**posix_trace_attr_getclockres, posix_trace_attr_getcreatetime,
posix_trace_attr_getgenversion, posix_trace_attr_getname, posix_trace_attr_setname**

Purpose: Retrieve and set information about a trace stream (**TRACING**).

OB TRC Synopsis:

```
#include <time.h>
#include <trace.h>

int posix_trace_attr_getclockres(const trace_attr_t *attr,
    struct timespec *resolution);
int posix_trace_attr_getcreatetime(const trace_attr_t *attr,
    struct timespec *createtime);

#include <trace.h>

int posix_trace_attr_getgenversion(const trace_attr_t *attr,
    char *genversion);
int posix_trace_attr_getname(const trace_attr_t *attr,
    char *tracename);
int posix_trace_attr_setname(trace_attr_t *attr,
    const char *tracename);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The `posix_trace_attr_getclockres()`, `posix_trace_attr_getcreatetime()`, `posix_trace_attr_getgenversion()`, `posix_trace_attr_getname()`, and `posix_trace_attr_setname()` functions are marked obsolescent.

**posix_trace_attr_getinherited, posix_trace_attr_getlogfullpolicy,
posix_trace_attr_getstreamfullpolicy, posix_trace_attr_setinherited,
posix_trace_attr_setlogfullpolicy, posix_trace_attr_setstreamfullpolicy**

Purpose: Retrieve and set the behavior of a trace stream (**TRACING**).

OB TRC Synopsis:

```
#include <trace.h>
```

TRI

```
int posix_trace_attr_getinherited(
    const trace_attr_t *restrict attr,
    int *restrict inheritancepolicy);
```

TRL

```
int posix_trace_attr_getlogfullpolicy(
    const trace_attr_t *restrict attr,
    int *restrict logpolicy);
```

```
int posix_trace_attr_getstreamfullpolicy(
    const trace_attr_t *restrict
    attr, int *restrict streampolicy);
```

TRI

```
int posix_trace_attr_setinherited(trace_attr_t *attr,
    int inheritancepolicy);
```

TRL

```
int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
    int logpolicy);
```

```
int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,
    int streampolicy);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: SD5-XSH-ERN-116 is applied, adding the missing **restrict** keyword to the `posix_trace_attr_getstreamfullpolicy()` function declaration.

These functions are marked obsolescent.

**posix_trace_attr_getlogsize, posix_trace_attr_getmaxdatasize,
 posix_trace_attr_getmaxsystemeventsize, posix_trace_attr_getmaxusereventsize,
 posix_trace_attr_getstreamsize, posix_trace_attr_setlogsize,
 posix_trace_attr_setmaxdatasize, posix_trace_attr_setstreamsize**

Purpose: Retrieve and set trace stream size attributes (**TRACING**).

```
OB TRC Synopsis: #include <sys/types.h>
                 #include <trace.h>

TRL             int posix_trace_attr_getlogsize(
                 const trace_attr_t *restrict attr,
                 size_t *restrict logsize);

                 int posix_trace_attr_getmaxdatasize(
                 const trace_attr_t *restrict attr,
                 size_t *restrict maxdatasize);

                 int posix_trace_attr_getmaxsystemeventsize(
                 const trace_attr_t *restrict attr,
                 size_t *restrict eventsize);

                 int posix_trace_attr_getmaxusereventsize(
                 const trace_attr_t *restrict attr,
                 size_t data_len, size_t *restrict eventsize);

                 int posix_trace_attr_getstreamsize(
                 const trace_attr_t *restrict attr,
                 size_t *restrict streamsize);

TRL             int posix_trace_attr_setlogsize(trace_attr_t *attr,
                 size_t logsize);

                 int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,
                 size_t maxdatasize);

                 int posix_trace_attr_setstreamsize(trace_attr_t *attr,
                 size_t streamsize);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: These functions are marked obsolescent.

posix_trace_clear

Purpose: Clear trace stream and trace log (**TRACING**).

```
OB TRC Synopsis: #include <sys/types.h>
                 #include <trace.h>

                 int posix_trace_clear(trace_id_t trid);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The *posix_trace_clear()* function is marked obsolescent.

posix_trace_close, posix_trace_open, posix_trace_rewind

Purpose: Trace log management (**TRACING**).

OB TRC Synopsis:

```
#include <trace.h>
```

TRL

```
int posix_trace_close(trace_id_t trid);
int posix_trace_open(int file_desc, trace_id_t *trid);
int posix_trace_rewind(trace_id_t trid);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The *posix_trace_close()*, *posix_trace_open()*, and *posix_trace_rewind()* functions are marked obsolescent.

posix_trace_create, posix_trace_create_withlog, posix_trace_flush, posix_trace_shutdown

Purpose: Trace stream initialization, flush, and shutdown from a process (**TRACING**).

OB TRC Synopsis:

```
#include <sys/types.h>
#include <trace.h>
```

TRL

```
int posix_trace_create(pid_t pid,
    const trace_attr_t *restrict attr,
    trace_id_t *restrict trid);
int posix_trace_create_withlog(pid_t pid,
    const trace_attr_t *restrict attr, int file_desc,
    trace_id_t *restrict trid);
int posix_trace_flush(trace_id_t trid);
int posix_trace_shutdown(trace_id_t trid);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: These functions are marked obsolescent.

SD5-XSH-ERN-154 is applied, updating the DESCRIPTION to remove the *posix_trace_trygetnext_event()* function from the list of functions that use the *trid* argument.

posix_trace_event, posix_trace_eventid_open

Purpose: Trace functions for instrumenting application code (**TRACING**).

OB TRC Synopsis:

```
#include <sys/types.h>
#include <trace.h>
```

```
void posix_trace_event(trace_event_id_t event_id,
    const void *restrict data_ptr, size_t data_len);
int posix_trace_eventid_open(const char *restrict event_name,
    trace_event_id_t *restrict event_id);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The *posix_trace_event()* and *posix_trace_eventid_open()* functions are marked obsolescent.

**posix_trace_eventid_equal, posix_trace_eventid_get_name,
posix_trace_trid_eventid_open**

Purpose: Manipulate the trace event type identifier (**TRACING**).

```
OB TRC Synopsis: #include <trace.h>
                int posix_trace_eventid_equal(trace_id_t trid,
                trace_event_id_t event1,
                trace_event_id_t event2);
                int posix_trace_eventid_get_name(trace_id_t trid,
                trace_event_id_t event, char *event_name);
TEF             int posix_trace_trid_eventid_open(trace_id_t trid,
                const char *restrict event_name,
                trace_event_id_t *restrict event);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: These functions are marked obsolescent.

**posix_trace_eventset_add, posix_trace_eventset_del, posix_trace_eventset_empty,
posix_trace_eventset_fill, posix_trace_eventset_ismember**

Purpose: Manipulate trace event type sets (**TRACING**).

```
OB TRC Synopsis: #include <trace.h>
TEF             int posix_trace_eventset_add(trace_event_id_t event_id,
                trace_event_set_t *set);
                int posix_trace_eventset_del(trace_event_id_t event_id,
                trace_event_set_t *set);
                int posix_trace_eventset_empty(trace_event_set_t *set);
                int posix_trace_eventset_fill(trace_event_set_t *set,
                int what);
                int posix_trace_eventset_ismember(trace_event_id_t event_id,
                const trace_event_set_t *restrict set,
                int *restrict ismember);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The `posix_trace_eventset_add()`, `posix_trace_eventset_del()`, `posix_trace_eventset_empty()`, `posix_trace_eventset_fill()`, and `posix_trace_eventset_ismember()` functions are marked obsolescent.

posix_trace_eventtypelist_getnext_id, posix_trace_eventtypelist_rewind

Purpose: Iterate over a mapping of trace event types (**TRACING**).

```
OB TRC Synopsis: #include <trace.h>
                int posix_trace_eventtypelist_getnext_id(trace_id_t trid,
                trace_event_id_t *restrict event,
                int *restrict unavailable);
                int posix_trace_eventtypelist_rewind(trace_id_t trid);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The `posix_trace_eventtypelist_getnext_id()` and `posix_trace_eventtypelist_rewind()` functions are marked obsolescent.

posix_trace_get_attr, posix_trace_get_status

Purpose: Retrieve the trace attributes or trace status (**TRACING**).

OB TRC Synopsis:

```
#include <trace.h>

int posix_trace_get_attr(trace_id_t trid, trace_attr_t *attr);
int posix_trace_get_status(trace_id_t trid,
    struct posix_trace_status_info *statusinfo);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The `posix_trace_get_attr()` and `posix_trace_get_status()` functions are marked obsolescent.

posix_trace_get_filter, posix_trace_set_filter

Purpose: Retrieve and set the filter of an initialized trace stream (**TRACING**).

OB TRC Synopsis:

```
#include <trace.h>

TEF int posix_trace_get_filter(trace_id_t trid,
    trace_event_set_t *set);
int posix_trace_set_filter(trace_id_t trid,
    const trace_event_set_t *set, int how);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The `posix_trace_get_filter()` and `posix_trace_set_filter()` functions are marked obsolescent.

posix_trace_getnext_event, posix_trace_timedgetnext_event, posix_trace_trygetnext_event

Purpose: Retrieve a trace event (**TRACING**).

OB TRC Synopsis:

```
#include <sys/types.h>
#include <trace.h>

int posix_trace_getnext_event(trace_id_t trid,
    struct posix_trace_event_info *restrict event,
    void *restrict data, size_t num_bytes,
    size_t *restrict data_len, int *restrict unavailable);
int posix_trace_timedgetnext_event(trace_id_t trid,
    struct posix_trace_event_info *restrict event,
    void *restrict data, size_t num_bytes,
    size_t *restrict data_len, int *restrict unavailable,
    const struct timespec *restrict abstime);
int posix_trace_trygetnext_event(trace_id_t trid,
    struct posix_trace_event_info *restrict event,
    void *restrict data, size_t num_bytes,
    size_t *restrict data_len, int *restrict unavailable);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The `posix_trace_getnext_event()`, `posix_trace_timedgetnext_event()`, and `posix_trace_trygetnext_event()` functions are marked obsolescent.

Functionality relating to the Timers option is moved to the Base.

posix_trace_start, posix_trace_stop

Purpose: Trace start and stop (**TRACING**).

OB TRC Synopsis:

```
#include <trace.h>

int posix_trace_start(trace_id_t trid);
int posix_trace_stop (trace_id_t trid);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: The `posix_trace_start()` and `posix_trace_stop()` functions are marked obsolescent.

posix_typed_mem_get_info

Purpose: Query typed memory information (**ADVANCED REALTIME**).

TYM Synopsis:

```
#include <sys/mman.h>

int posix_typed_mem_get_info(int fildes,
    struct posix_typed_mem_info *info);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: No functional changes are made in this issue.

posix_typed_mem_open

Purpose: Open a typed memory object (**ADVANCED REALTIME**).

TYM Synopsis:

```
#include <sys/mman.h>

int posix_typed_mem_open(const char *name, int oflag,
    int tflag);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

pow, powf, powl

Purpose: Power function.

Synopsis:

```
#include <math.h>

double pow(double x, double y);
float powf(float x, float y);
long double powl(long double x, long double y);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #51 (SD5-XSH-ERN-81) is applied.

pselect, select

Purpose: Synchronous I/O multiplexing.

Synopsis: `#include <sys/select.h>`

```
int pselect(int nfds, fd_set *restrict readfds,
            fd_set *restrict writefds, fd_set *restrict errorfds,
            const struct timespec *restrict timeout,
            const sigset_t *restrict sigmask);
int select(int nfds, fd_set *restrict readfds,
           fd_set *restrict writefds, fd_set *restrict errorfds,
           struct timeval *restrict timeout);
void FD_CLR(int fd, fd_set *fdset);
int FD_ISSET(int fd, fd_set *fdset);
void FD_SET(int fd, fd_set *fdset);
void FD_ZERO(fd_set *fdset);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: SD5-XSH-ERN-122 is applied, adding text to the DESCRIPTION for when a thread is canceled during a call to *pselect()*, and adding example code to the RATIONALE.

Functionality relating to the XSI STREAMS option is marked obsolescent.

Functionality relating to the Threads option is moved to the Base.

psiginfo, psignal

Purpose: Print signal information to standard error.

cx Synopsis: `#include <signal.h>`

```
void psiginfo(const siginfo_t *pinfo, const char *message);
void psignal(int signum, const char *message);
```

The *psiginfo()* and *psignal()* functions print a message out on *stderr* associated with a signal number.

Application writers should note that System V historically has *psignal()* and *psiginfo()* in **<siginfo.h>**. However, the **<siginfo.h>** header is not specified in the Base Definitions volume of POSIX.1-2008, and the type **siginfo_t** is defined in **<signal.h>**.

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Issue 7: First released in Issue 7.

pthread_atfork

Purpose: Register fork handlers.

Synopsis: `#include <pthread.h>`

```
int pthread_atfork(void (*prepare)(void), void (*parent)(void),
                  void (*child)(void));
```

Derivation: First released in Issue 5. Derived from the POSIX Threads Extension.

Issue 7: The `pthread_atfork()` function is moved from the Threads option to the Base.

SD5-XSH-ERN-145 is applied, updating the RATIONALE to confirm the requirement that a child of a multi-threaded process may only execute async-signal-safe operations until such time as one of the `exec` functions is called.

pthread_attr_destroy, pthread_attr_init

Purpose: Destroy and initialize the thread attributes object.

Synopsis: `#include <pthread.h>`

```
int pthread_attr_destroy(pthread_attr_t *attr);
int pthread_attr_init(pthread_attr_t *attr);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_attr_destroy()` and `pthread_attr_init()` functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

The [EBUSY] error for an already initialized thread attributes object is removed; this condition results in undefined behavior.

pthread_attr_getdetachstate, pthread_attr_setdetachstate

Purpose: Get and set the detachstate attribute.

Synopsis: `#include <pthread.h>`

```
int pthread_attr_getdetachstate(const pthread_attr_t *attr,
                               int *detachstate);
int pthread_attr_setdetachstate(pthread_attr_t *attr,
                               int detachstate);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_attr_setdetachstate()` and `pthread_attr_getdetachstate()` functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

pthread_attr_getguardsize, pthread_attr_setguardsize

Purpose: Get and set the thread guardsize attribute.

Synopsis: `#include <pthread.h>`

```
int pthread_attr_getguardsize(
    const pthread_attr_t *restrict attr,
    size_t *restrict guardsize);
int pthread_attr_setguardsize(pthread_attr_t *attr,
    size_t guardsize);
```

Derivation: First released in Issue 5.

Issue 7: SD5-XSH-ERN-111 is applied, removing the reference to the *stack* attribute in the DESCRIPTION.

SD5-XSH-ERN-175 is applied, updating the DESCRIPTION to note that the default size of the guard area is implementation-defined.

The *pthread_attr_getguardsize()* and *pthread_attr_setguardsize()* functions are moved from the XSI option to the Base.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

pthread_attr_getinheritsched, pthread_attr_setinheritsched

Purpose: Get and set the inheritsched attribute (**REALTIME THREADS**).

TPS

Synopsis: `#include <pthread.h>`

```
int pthread_attr_getinheritsched(
    const pthread_attr_t *restrict attr,
    int *restrict inheritsched);
int pthread_attr_setinheritsched(pthread_attr_t *attr,
    int inheritsched);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The *pthread_attr_getinheritsched()* and *pthread_attr_setinheritsched()* functions are moved from the Threads option.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

pthread_attr_getschedparam, pthread_attr_setschedparam

Purpose: Get and set the schedparam attribute.

Synopsis: `#include <pthread.h>`

```
int pthread_attr_getschedparam(
    const pthread_attr_t *restrict attr,
    struct sched_param *restrict param);
int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
    const struct sched_param *restrict param);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_attr_getschedparam()` and `pthread_attr_setschedparam()` functions are moved from the Threads option to the Base.

Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements for the `sched_ss_repl_period` and `sched_ss_init_budget` values.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

pthread_attr_getschedpolicy, pthread_attr_setschedpolicy

Purpose: Get and set the schedpolicy attribute (**REALTIME THREADS**).

TPS

Synopsis:

```
#include <pthread.h>

int pthread_attr_getschedpolicy(
    const pthread_attr_t *restrict attr,
    int *restrict policy);
int pthread_attr_setschedpolicy(pthread_attr_t *attr,
    int policy);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_attr_getschedpolicy()` and `pthread_attr_setschedpolicy()` functions are moved from the Threads option.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

pthread_attr_getscope, pthread_attr_setscope

Purpose: Get and set the contentionscope attribute (**REALTIME THREADS**).

TPS

Synopsis:

```
#include <pthread.h>

int pthread_attr_getscope(const pthread_attr_t *restrict attr,
    int *restrict contentionscope);
int pthread_attr_setscope(pthread_attr_t *attr,
    int contentionscope);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_attr_getscope()` and `pthread_attr_setscope()` functions are moved from the Threads option.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

pthread_attr_getstack, pthread_attr_setstack

Purpose: Get and set stack attributes.

TSA TSS Synopsis:

```
#include <pthread.h>

int pthread_attr_getstack(const pthread_attr_t *restrict attr,
    void **restrict stackaddr, size_t *restrict stacksize);
int pthread_attr_setstack(pthread_attr_t *attr,
    void *stackaddr, size_t stacksize);
```

Derivation: First released in Issue 6.

Issue 7: SD5-XSH-ERN-66 is applied, correcting the use of *attr* in the [EINVAL] error condition.

Austin Group Interpretation 1003.1-2001 #057 is applied, clarifying the behavior if the function is called before the *stackaddr* attribute is set.

SD5-XSH-ERN-157 is applied, updating the APPLICATION USAGE section.

The description of the *stackaddr* attribute is updated in the DESCRIPTION and APPLICATION USAGE sections.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

pthread_attr_getstacksize, pthread_attr_setstacksize

Purpose: Get and set the stacksize attribute.

TSS Synopsis:

```
#include <pthread.h>

int pthread_attr_getstacksize(
    const pthread_attr_t *restrict attr,
    size_t *restrict stacksize);
int pthread_attr_setstacksize(pthread_attr_t *attr,
    size_t stacksize);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The *pthread_attr_getstacksize()* and *pthread_attr_setstacksize()* functions are moved from the Threads option.

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

pthread_barrier_destroy, pthread_barrier_init

Purpose: Destroy and initialize a barrier object.

Synopsis:

```
#include <pthread.h>

int pthread_barrier_destroy(pthread_barrier_t *barrier);
int pthread_barrier_init(pthread_barrier_t *restrict barrier,
    const pthread_barrierattr_t *restrict attr,
    unsigned count);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The `pthread_barrier_destroy()` and `pthread_barrier_init()` functions are moved from the Barriers option to the Base.

The [EINVAL] error for an uninitialized barrier object and an uninitialized barrier attributes object is removed; this condition results in undefined behavior.

The [EBUSY] error for a barrier that is in use or an already initialized barrier object is removed; this condition results in undefined behavior.

pthread_barrier_wait

Purpose: Synchronize at a barrier.

Synopsis: `#include <pthread.h>`

```
int pthread_barrier_wait(pthread_barrier_t *barrier);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The `pthread_barrier_wait()` function is moved from the Barriers option to the Base.

The [EINVAL] error for an uninitialized barrier object is removed; this condition results in undefined behavior.

pthread_barrierattr_destroy, pthread_barrierattr_init

Purpose: Destroy and initialize the barrier attributes object.

Synopsis: `#include <pthread.h>`

```
int pthread_barrierattr_destroy(pthread_barrierattr_t *attr);
int pthread_barrierattr_init(pthread_barrierattr_t *attr);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The `pthread_barrierattr_destroy()` and `pthread_barrierattr_init()` functions are moved from the Barriers option to the Base.

The [EINVAL] error for an uninitialized barrier attributes object is removed; this condition results in undefined behavior.

pthread_barrierattr_getpshared, pthread_barrierattr_setpshared

Purpose: Get and set the process-shared attribute of the barrier attributes object.

TSH Synopsis: `#include <pthread.h>`

```
int pthread_barrierattr_getpshared(
    const pthread_barrierattr_t *restrict attr,
    int *restrict pshared);
int pthread_barrierattr_setpshared(
    pthread_barrierattr_t *attr, int pshared);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000

Issue 7: The `pthread_barrierattr_getpshared()` and `pthread_barrierattr_setpshared()` functions are moved from the Barriers option.

The [EINVAL] error for an uninitialized barrier attributes object is removed; this condition results in undefined behavior.

pthread_cancel

Purpose: Cancel execution of a thread.

Synopsis: `#include <pthread.h>`
`int pthread_cancel(pthread_t thread);`

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_cancel()` function is moved from the Threads option to the Base. Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

pthread_cleanup_pop, pthread_cleanup_push

Purpose: Establish cancellation handlers.

Synopsis: `#include <pthread.h>`
`void pthread_cleanup_pop(int execute);`
`void pthread_cleanup_push(void (*routine)(void*), void *arg);`

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_cleanup_pop()` and `pthread_cleanup_push()` functions are moved from the Threads option to the Base.

pthread_cond_broadcast, pthread_cond_signal

Purpose: Broadcast or signal a condition.

Synopsis: `#include <pthread.h>`
`int pthread_cond_broadcast(pthread_cond_t *cond);`
`int pthread_cond_signal(pthread_cond_t *cond);`

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_cond_broadcast()` and `pthread_cond_signal()` functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized condition variable is removed; this condition results in undefined behavior.

pthread_cond_destroy, pthread_cond_init

Purpose: Destroy and initialize condition variables.

Synopsis: `#include <pthread.h>`
`int pthread_cond_destroy(pthread_cond_t *cond);`
`int pthread_cond_init(pthread_cond_t *restrict cond,`
`const pthread_condattr_t *restrict attr);`
`pthread_cond_t cond = PTHREAD_COND_INITIALIZER;`

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_cond_destroy()` and `pthread_cond_init()` functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized condition variable and an uninitialized condition variable attributes object is removed; this condition results in undefined behavior.

The [EBUSY] error for a condition variable already in use or an already initialized condition variable is removed; this condition results in undefined behavior.

pthread_cond_timedwait, pthread_cond_wait

Purpose: Wait on a condition.

Synopsis: `#include <pthread.h>`

```
int pthread_cond_timedwait(pthread_cond_t *restrict cond,
    pthread_mutex_t *restrict mutex,
    const struct timespec *restrict abstime);
int pthread_cond_wait(pthread_cond_t *restrict cond,
    pthread_mutex_t *restrict mutex);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: SD5-XSH-ERN-44 is applied, changing the definition of the “shall fail” case of the [EINVAL] error.

Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

The `pthread_cond_timedwait()` and `pthread_cond_wait()` functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized condition variable or uninitialized mutex object is removed; this condition results in undefined behavior”

The [EPERM] error is revised and moved to the “shall fail” list of error conditions for the `pthread_cond_timedwait()` function.

The DESCRIPTION is updated to clarify the behavior when `mutex` is a robust mutex.

The ERRORS section is updated to include “shall fail” cases for PTHREAD_MUTEX_ERRORCHECK mutexes.

The DESCRIPTION is rewritten to clarify that undefined behavior occurs only for mutexes where the [EPERM] error is not mandated.

pthread_condattr_destroy, pthread_condattr_init

Purpose: Destroy and initialize the condition variable attributes object.

Synopsis: `#include <pthread.h>`

```
int pthread_condattr_destroy(pthread_condattr_t *attr);
int pthread_condattr_init(pthread_condattr_t *attr);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_condattr_destroy()` and `pthread_condattr_init()` functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized condition variable attributes object is removed; this condition results in undefined behavior.

pthread_condattr_getclock, pthread_condattr_setclock

Purpose: Get and set the clock selection condition variable attribute.

Synopsis: `#include <pthread.h>`

```
int pthread_condattr_getclock(
    const pthread_condattr_t *restrict attr,
    clockid_t *restrict clock_id);
int pthread_condattr_setclock(pthread_condattr_t *attr,
    clockid_t clock_id);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The `pthread_condattr_getclock()` and `pthread_condattr_setclock()` functions are moved from the Clock Selection option to the Base.

The [EINVAL] error for an uninitialized condition variable attributes object is removed; this condition results in undefined behavior.

pthread_condattr_getpshared, pthread_condattr_setpshared

Purpose: Get and set the process-shared condition variable attributes.

TSH

Synopsis: `#include <pthread.h>`

```
int pthread_condattr_getpshared(
    const pthread_condattr_t *restrict attr,
    int *restrict pshared);
int pthread_condattr_setpshared(pthread_condattr_t *attr,
    int pshared);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_condattr_getpshared()` and `pthread_condattr_setpshared()` functions are moved from the Threads option.

The [EINVAL] error for an uninitialized condition variable attributes object is removed; this condition results in undefined behavior.

pthread_create

Purpose: Thread creation.

Synopsis: `#include <pthread.h>`

```
int pthread_create(pthread_t *restrict thread,
    const pthread_attr_t *restrict attr,
    void *(*start_routine)(void*), void *restrict arg);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_create()` function is moved from the Threads option to the Base.
 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

pthread_detach

Purpose: Detach a thread.

Synopsis:

```
#include <pthread.h>
int pthread_detach(pthread_t thread);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_detach()` function is moved from the Threads option to the Base.
 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.
 The [EINVAL] error for a non-joinable thread is removed; this condition results in undefined behavior.

pthread_equal

Purpose: Compare thread IDs.

Synopsis:

```
#include <pthread.h>
int pthread_equal(pthread_t t1, pthread_t t2);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_equal()` function is moved from the Threads option to the Base.

pthread_exit

Purpose: Thread termination.

Synopsis:

```
#include <pthread.h>
void pthread_exit(void *value_ptr);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_exit()` function is moved from the Threads option to the Base.

pthread_getconcurrency, pthread_setconcurrency

Purpose: Get and set the level of concurrency.

OB XSI Synopsis:

```
#include <pthread.h>
int pthread_getconcurrency(void);
int pthread_setconcurrency(int new_level);
```

Derivation: First released in Issue 5.

Issue 7: The `pthread_getconcurrency()` and `pthread_setconcurrency()` functions are marked obsolescent.

pthread_getcpuclockid

Purpose: Access a thread CPU-time clock (**ADVANCED REALTIME THREADS**).

TCT

Synopsis:

```
#include <pthread.h>
#include <time.h>

int pthread_getcpuclockid(pthread_t thread_id,
    clockid_t *clock_id);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: The `pthread_getcpuclockid()` function is moved from the Threads option. Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

pthread_getschedparam, pthread_setschedparam

Purpose: Dynamic thread scheduling parameters access (**REALTIME THREADS**).

TPS

Synopsis:

```
#include <pthread.h>

int pthread_getschedparam(pthread_t thread,
    int *restrict policy,
    struct sched_param *restrict param);
int pthread_setschedparam(pthread_t thread, int policy,
    const struct sched_param *param);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_getschedparam()` and `pthread_setschedparam()` functions are moved from the Threads option. Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements for the `sched_ss_repl_period` and `sched_ss_init_budget` values. Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

pthread_getspecific, pthread_setspecific

Purpose: Thread-specific data management.

Synopsis:

```
#include <pthread.h>

void *pthread_getspecific(pthread_key_t key);
int pthread_setspecific(pthread_key_t key, const void *value);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_getspecific()` and `pthread_setspecific()` functions are moved from the Threads option to the Base.

The [EINVAL] error for a key value not obtained from `pthread_key_create()` or a

key deleted with *pthread_key_delete()* is removed; this condition results in undefined behavior.

pthread_join

Purpose: Wait for thread termination.

Synopsis: `#include <pthread.h>`

```
int pthread_join(pthread_t thread, void **value_ptr);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The *pthread_join()* function is moved from the Threads option to the Base.

Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

The [EINVAL] error for a non-joinable thread is removed; this condition results in undefined behavior.

The [EDEADLK] error for the calling thread is removed; this condition results in undefined behavior.

pthread_key_create

Purpose: Thread-specific data key creation.

Synopsis: `#include <pthread.h>`

```
int pthread_key_create(pthread_key_t *key,
    void (*destructor)(void*));
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The *pthread_key_create()* function is moved from the Threads option to the Base.

pthread_key_delete

Purpose: Thread-specific data key deletion.

Synopsis: `#include <pthread.h>`

```
int pthread_key_delete(pthread_key_t key);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The *pthread_key_delete()* function is moved from the Threads option to the Base.

The [EINVAL] error for a key value not obtained from *pthread_key_create()* or a key deleted with *pthread_key_delete()* is removed; this condition results in undefined behavior.

pthread_kill

Purpose: Send a signal to a thread.

cx

Synopsis:

```
#include <signal.h>
int pthread_kill(pthread_t thread, int sig);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The *pthread_kill()* function is moved from the Threads option to the Base. Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

pthread_mutex_consistent

Purpose: Mark state protected by robust mutex as consistent.

Synopsis:

```
#include <pthread.h>
int pthread_mutex_consistent(pthread_mutex_t *mutex);
```

If *mutex* is a robust mutex in an inconsistent state, the *pthread_mutex_consistent()* function can be used to mark the state protected by the mutex referenced by *mutex* as consistent again.

Application writers should note that the *pthread_mutex_consistent()* function is only responsible for notifying the implementation that the state protected by the mutex has been recovered and that normal operations with the mutex can be resumed. It is the responsibility of the application to recover the state so it can be reused. If the application is not able to perform the recovery, it can notify the implementation that the situation is unrecoverable by a call to *pthread_mutex_unlock()* without a prior call to *pthread_mutex_consistent()*, in which case subsequent threads that attempt to lock the mutex will fail to acquire the lock and be returned [ENOTRECOVERABLE].

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 3.

Issue 7: First released in Issue 7.

pthread_mutex_destroy, pthread_mutex_init

Purpose: Destroy and initialize a mutex.

Synopsis:

```
#include <pthread.h>
int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_mutex_init(pthread_mutex_t *restrict mutex,
    const pthread_mutexattr_t *restrict attr);
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

The *pthread_mutex_destroy()* and *pthread_mutex_init()* functions are moved from

the Threads option to the Base.

The [EINVAL] error for an uninitialized mutex or an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

The [EBUSY] error for a locked mutex, a mutex that is referenced, or an already initialized mutex is removed; this condition results in undefined behavior.

pthread_mutex_getprioceiling, pthread_mutex_setprioceiling

Purpose: Get and set the priority ceiling of a mutex (**REALTIME THREADS**).

RPP|TPP

Synopsis:

```
#include <pthread.h>

int pthread_mutex_getprioceiling(
    const pthread_mutex_t *restrict mutex,
    int *restrict prioceiling);
int pthread_mutex_setprioceiling(
    pthread_mutex_t *restrict mutex,
    int prioceiling, int *restrict old_ceiling);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: Austin Group Interpretation 1003.1-2001 #052 is applied, adding [EDEADLK] as a “may fail” error.

SD5-XSH-ERN-158 is applied, updating the ERRORS section to include a “shall fail” error case for when the protocol attribute of *mutex* is PTHREAD_PRIO_NONE.

The *pthread_mutex_getprioceiling()* and *pthread_mutex_setprioceiling()* functions are moved from the Threads option to require support of either the Robust Mutex Priority Protection option or the Non-Robust Mutex Priority Protection option.

The DESCRIPTION and ERRORS sections are updated to account properly for all of the various mutex types.

pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock

Purpose: Lock and unlock a mutex.

Synopsis:

```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: SD5-XSH-ERN-43 is applied, marking the “shall fail” case of the [EINVAL] error as dependent on the Thread Priority Protection option.

Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

The *pthread_mutex_lock()*, *pthread_mutex_trylock()*, and *pthread_mutex_unlock()* functions are moved from the Threads option to the Base.

The PTHREAD_MUTEX_NORMAL, PTHREAD_MUTEX_ERRORCHECK, PTHREAD_MUTEX_RECURSIVE, and PTHREAD_MUTEX_DEFAULT extended

mutex types are moved from the XSI option to the Base.

The DESCRIPTION is updated to clarify the behavior when *mutex* does not refer to an initialized mutex.

The ERRORS section is updated to account properly for all of the various mutex types.

pthread_mutex_timedlock

Purpose: Lock a mutex.

Synopsis:

```
#include <pthread.h>
#include <time.h>

int pthread_mutex_timedlock(pthread_mutex_t *restrict mutex,
    const struct timespec *restrict abstime);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

The *pthread_mutex_timedlock()* function is moved from the Timeouts option to the Base.

Functionality relating to the Timers option is moved to the Base.

The DESCRIPTION is updated to clarify the behavior when *mutex* does not refer to an initialized mutex.

The ERRORS section is updated to account properly for all of the various mutex types.

pthread_mutexattr_destroy, pthread_mutexattr_init

Purpose: Destroy and initialize the mutex attributes object.

Synopsis:

```
#include <pthread.h>

int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
int pthread_mutexattr_init(pthread_mutexattr_t *attr);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The *pthread_mutexattr_destroy()* and *pthread_mutexattr_init()* functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

pthread_mutexattr_getprioceiling, pthread_mutexattr_setprioceiling

Purpose: Get and set the prioceiling attribute of the mutex attributes object (**REALTIME THREADS**).

RPP|TPP Synopsis:

```
#include <pthread.h>

int pthread_mutexattr_getprioceiling(
    const pthread_mutexattr_t *restrict attr,
    int *restrict prioceiling);
int pthread_mutexattr_setprioceiling(
```

```
pthread_mutexattr_t *attr, int prioceiling);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_mutexattr_getprioceiling()` and `pthread_mutexattr_setprioceiling()` functions are moved from the Threads option to require support of either the Robust Mutex Priority Protection option or the Non-Robust Mutex Priority Protection option.

The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

pthread_mutexattr_getprotocol, pthread_mutexattr_setprotocol

Purpose: Get and set the protocol attribute of the mutex attributes object (**REALTIME THREADS**).

MC1 Synopsis:

```
#include <pthread.h>

int pthread_mutexattr_getprotocol(const pthread_mutexattr_t
    *restrict attr, int *restrict protocol);
int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
    int protocol);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: SD5-XSH-ERN-135 is applied, updating the DESCRIPTION to define a default value for the `protocol` attribute.

SD5-XSH-ERN-188 is applied, clarifying that propagation of the priority inheritance effect only applies if the other mutex has the protocol attribute `PTHREAD_PRIO_INHERIT`.

The `pthread_mutexattr_getprotocol()` and `pthread_mutexattr_setprotocol()` functions are moved from the Threads option to require support of either the Non-Robust Mutex Priority Protection option or the Non-Robust Mutex Priority Inheritance option or the Robust Mutex Priority Protection option or the Robust Mutex Priority Inheritance option.

The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

pthread_mutexattr_getpshared, pthread_mutexattr_setpshared

Purpose: Get and set the process-shared attribute.

TSH Synopsis:

```
#include <pthread.h>

int pthread_mutexattr_getpshared(const pthread_mutexattr_t
    *restrict attr, int *restrict pshared);
int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
    int pshared);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_mutexattr_getpshared()` and `pthread_mutexattr_setpshared()` functions are moved from the Threads option.

The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

pthread_mutexattr_getrobust, pthread_mutexattr_setrobust

Purpose: Get and set the mutex robust attribute.

Synopsis: `#include <pthread.h>`

```
int pthread_mutexattr_getrobust(
    const pthread_mutexattr_t *restrict attr,
    int *restrict robust);
int pthread_mutexattr_setrobust(pthread_mutexattr_t *attr,
    int robust);
```

The `pthread_mutexattr_getrobust()` and `pthread_mutexattr_setrobust()` functions, respectively, get and set the mutex *robust* attribute.

Valid values for *robust* include:

PTHREAD_MUTEX_STALLED

No special actions are taken if the owner of the mutex is terminated while holding the mutex lock. This can lead to deadlocks if no other thread can unlock the mutex.

This is the default value.

PTHREAD_MUTEX_ROBUST

If the process containing the owning thread of a robust mutex terminates while holding the mutex lock, the next thread that acquires the mutex is notified about the termination by the return value [EOWNERDEAD] from the locking function. If the owning thread of a robust mutex terminates while holding the mutex lock, the next thread that acquires the mutex may be notified about the termination by the return value [EOWNERDEAD]. The notified thread can then attempt to mark the state protected by the mutex as consistent again by a call to `pthread_mutex_consistent()`. After a subsequent successful call to `pthread_mutex_unlock()`, the mutex lock is released and can be used normally by other threads. If the mutex is unlocked without a call to `pthread_mutex_consistent()`, it is placed in a permanently unusable state and all attempts to lock the mutex fail with the error [ENOTRECOVERABLE]. The only permissible operation on such a mutex is `pthread_mutex_destroy()`.

Application writers should note that the actions required to make the state protected by the mutex consistent again are solely dependent on the application. If it is not possible to make the state of a mutex consistent, robust mutexes can be used to notify this situation by calling `pthread_mutex_unlock()` without a prior call to `pthread_mutex_consistent()`.

If the state is declared inconsistent by calling `pthread_mutex_unlock()` without a prior call to `pthread_mutex_consistent()`, a possible approach could be to destroy the mutex and then reinitialize it. However, it should be noted that this is possible only in certain situations where the state protected by the mutex has to be reinitialized and coordination achieved with other threads blocked on the mutex,

because otherwise a call to a locking function with a reference to a mutex object invalidated by a call to `pthread_mutex_destroy()` results in undefined behavior.

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 3.

Issue 7: First released in Issue 7.

pthread_mutexattr_gettype, pthread_mutexattr_settype

Purpose: Get and set the mutex type attribute.

Synopsis:

```
#include <pthread.h>

int pthread_mutexattr_gettype(
    const pthread_mutexattr_t *restrict attr,
    int *restrict type);
int pthread_mutexattr_settype(pthread_mutexattr_t *attr,
    int type);
```

Derivation: First released in Issue 5.

Issue 7: The `pthread_mutexattr_gettype()` and `pthread_mutexattr_settype()` functions are moved from the XSI option to the Base.

The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

pthread_once

Purpose: Dynamic package initialization.

Synopsis:

```
#include <pthread.h>

int pthread_once(pthread_once_t *once_control,
    void (*init_routine)(void));
pthread_once_t once_control = PTHREAD_ONCE_INIT;
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_once()` function is moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized `pthread_once_t` object is removed; this condition results in undefined behavior.

pthread_rwlock_destroy, pthread_rwlock_init

Purpose: Destroy and initialize a read-write lock object.

Synopsis:

```
#include <pthread.h>

int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);
int pthread_rwlock_init(pthread_rwlock_t *restrict rwlock,
    const pthread_rwlockattr_t *restrict attr);
pthread_rwlock_t rwlock = PTHREAD_RWLOCK_INITIALIZER;
```

XSI

Derivation: First released in Issue 5.

Issue 7: Austin Group Interpretation 1003.1-2001 #048 is applied, adding the `PTHREAD_RWLOCK_INITIALIZER` macro.

The `pthread_rwlock_destroy()` and `pthread_rwlock_init()` functions are moved from the Threads option to the Base.

The `[EINVAL]` error for an uninitialized read-write lock object or read-write lock attributes object is removed; this condition results in undefined behavior.

The `[EBUSY]` error for a locked read-write lock object or an already initialized read-write lock object is removed; this condition results in undefined behavior.

pthread_rwlock_rdlock, pthread_rwlock_tryrdlock

Purpose: Lock a read-write lock object for reading.

Synopsis: `#include <pthread.h>`

```
int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);
```

Derivation: First released in Issue 5.

Issue 7: The `pthread_rwlock_rdlock()` and `pthread_rwlock_tryrdlock()` functions are moved from the Threads option to the Base.

The `[EINVAL]` error for an uninitialized read-write lock object is removed; this condition results in undefined behavior.

pthread_rwlock_timedrdlock

Purpose: Lock a read-write lock for reading.

Synopsis: `#include <pthread.h>`
`#include <time.h>`

```
int pthread_rwlock_timedrdlock(
    pthread_rwlock_t *restrict rwlock,
    const struct timespec *restrict abstime);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The `pthread_rwlock_timedrdlock()` function is moved from the Timeouts option to the Base.

The `[EINVAL]` error for an uninitialized read-write lock object is removed; this condition results in undefined behavior.

pthread_rwlock_timedwrlock

Purpose: Lock a read-write lock for writing.

Synopsis: `#include <pthread.h>`
`#include <time.h>`

```
int pthread_rwlock_timedwrlock(
    pthread_rwlock_t *restrict rwlock,
    const struct timespec *restrict abstime);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The `pthread_rwlock_timedwrlock()` function is moved from the Timeouts option to the Base.

The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results in undefined behavior.

pthread_rwlock_trywrlock, pthread_rwlock_wrlock

Purpose: Lock a read-write lock object for writing.

Synopsis: `#include <pthread.h>`

```
int pthread_rwlock_trywrlock(pthread_rwlock_t *rlock);
int pthread_rwlock_wrlock(pthread_rwlock_t *rlock);
```

Derivation: First released in Issue 5.

Issue 7: The `pthread_rwlock_trywrlock()` and `pthread_rwlock_wrlock()` functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results in undefined behavior.

pthread_rwlock_unlock

Purpose: Unlock a read-write lock object.

Synopsis: `#include <pthread.h>`

```
int pthread_rwlock_unlock(pthread_rwlock_t *rlock);
```

Derivation: First released in Issue 5.

Issue 7: The `pthread_rwlock_unlock()` function is moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results in undefined behavior.

The [EPERM] error for a read-write lock object for which the current thread does not hold a lock is removed; this condition results in undefined behavior.

pthread_rwlockattr_destroy, pthread_rwlockattr_init

Purpose: Destroy and initialize the read-write lock attributes object.

Synopsis: `#include <pthread.h>`

```
int pthread_rwlockattr_destroy(pthread_rwlockattr_t *attr);
int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);
```

Derivation: First released in Issue 5.

Issue 7: The `pthread_rwlockattr_destroy()` and `pthread_rwlockattr_init()` functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized read-write lock attributes object is removed; this condition results in undefined behavior.

pthread_rwlockattr_getpshared, pthread_rwlockattr_setpshared

Purpose: Get and set the process-shared attribute of the read-write lock attributes object.

TSH

```
Synopsis: #include <pthread.h>

int pthread_rwlockattr_getpshared(const pthread_rwlockattr_t
    *restrict attr, int *restrict pshared);
int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
    int pshared);
```

Derivation: First released in Issue 5.

Issue 7: The `pthread_rwlockattr_getpshared()` and `pthread_rwlockattr_setpshared()` functions are moved from the Threads option.

The [EINVAL] error for an uninitialized read-write lock attributes object is removed; this condition results in undefined behavior.

pthread_self

Purpose: Get the calling thread ID.

```
Synopsis: #include <pthread.h>

pthread_t pthread_self(void);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: Austin Group Interpretation 1003.1-2001 #063 is applied, updating the RETURN VALUE section to indicate that the `pthread_self()` function is always successful and no return value is reserved to indicate an error.

The `pthread_self()` function is moved from the Threads option to the Base.

pthread_setcancelstate, pthread_setcanceltype, pthread_testcancel

Purpose: Set cancelability state.

```
Synopsis: #include <pthread.h>

int pthread_setcancelstate(int state, int *oldstate);
int pthread_setcanceltype(int type, int *oldtype);
void pthread_testcancel(void);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: The `pthread_setcancelstate()`, `pthread_setcanceltype()`, and `pthread_testcancel()` functions are moved from the Threads option to the Base.

pthread_setschedprio

Purpose: Dynamic thread scheduling parameters access (**REALTIME THREADS**).

TPS

Synopsis:

```
#include <pthread.h>
int pthread_setschedprio(pthread_t thread, int prio);
```

Derivation: First released in Issue 6. Included as a response to IEEE PASC Interpretation 1003.1 #96.

Issue 7: The *pthread_setschedprio()* function is moved from the Threads option. Austin Group Interpretation 1003.1-2001 #069 is applied, updating the [EPERM] error. Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

pthread_sigmask, sigprocmask

Purpose: Examine and change blocked signals.

CX

Synopsis:

```
#include <signal.h>
int pthread_sigmask(int how, const sigset_t *restrict set,
sigset_t *restrict oset);
int sigprocmask(int how, const sigset_t *restrict set,
sigset_t *restrict oset);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: The *pthread_sigmask()* function is moved from the Threads option to the Base.

pthread_spin_destroy, pthread_spin_init

Purpose: Destroy or initialize a spin lock object.

Synopsis:

```
#include <pthread.h>
int pthread_spin_destroy(pthread_spinlock_t *lock);
int pthread_spin_init(pthread_spinlock_t *lock, int pshared);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The *pthread_spin_destroy()* and *pthread_spin_init()* functions are moved from the Spin Locks option to the Base.

The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in undefined behavior.

The [EBUSY] error for a locked spin lock object or an already initialized spin lock object is removed; this condition results in undefined behavior.

pthread_spin_lock, pthread_spin_trylock

Purpose: Lock a spin lock object.

Synopsis: `#include <pthread.h>`

```
int pthread_spin_lock(pthread_spinlock_t *lock);
int pthread_spin_trylock(pthread_spinlock_t *lock);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The `pthread_spin_lock()` and `pthread_spin_trylock()` functions are moved from the Spin Locks option to the Base.

The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in undefined behavior.

The [EDEADLK] error for a spin lock object for which the calling thread already holds the lock is removed; this condition results in undefined behavior.

pthread_spin_unlock

Purpose: Unlock a spin lock object.

Synopsis: `#include <pthread.h>`

```
int pthread_spin_unlock(pthread_spinlock_t *lock);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7: The `pthread_spin_unlock()` function is moved from the Spin Locks option to the Base.

The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in undefined behavior.

The [EPERM] error for a spin lock object for which the current thread does not hold the lock is removed; this condition results in undefined behavior.

ptsname

Purpose: Get name of the slave pseudo-terminal device.

XS1 Synopsis: `#include <stdlib.h>`
`char *ptsname(int fildes);`

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

putc

Purpose: Put a byte on a stream.

Synopsis: `#include <stdio.h>`

```
int putc(int c, FILE *stream);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

putchar

Purpose: Put a byte on a stdout stream.

Synopsis: `#include <stdio.h>`
`int putchar(int c);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

putenv

Purpose: Change or add a value to an environment.

XSI Synopsis: `#include <stdlib.h>`
`int putenv(char *string);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

putmsg, putpmsg

Purpose: Send a message on a STREAM (**STREAMS**).

OB XSR Synopsis: `#include <stropts.h>`
`int putmsg(int fildes, const struct strbuf *ctlptr,`
`const struct strbuf *dataptr, int flags);`
`int putpmsg(int fildes, const struct strbuf *ctlptr,`
`const struct strbuf *dataptr, int band, int flags);`

Derivation: First released in Issue 4, Version 2.

Issue 7: The `putmsg()` and `putpmsg()` functions are marked obsolescent.

puts

Purpose: Put a string on standard output.

Synopsis: `#include <stdio.h>`
`int puts(const char *s);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Changes are made related to support for finegrained timestamps.

putwc

Purpose: Put a wide character on a stream.

Synopsis: `#include <stdio.h>`
`#include <wchar.h>`
`wint_t putwc(wchar_t wc, FILE *stream);`

Derivation: First released as a World-wide Portability Interface in Issue 4.

Issue 7: No functional changes are made in this issue.

putwchar

Purpose: Put a wide character on a stdout stream.

Synopsis: `#include <wchar.h>`
`wint_t putwchar(wchar_t wc);`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

qsort

Purpose: Sort a table of data.

Synopsis: `#include <stdlib.h>`
`void qsort(void *base, size_t nel, size_t width,
int (*compar)(const void *, const void *));`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

raise

Purpose: Send a signal to the executing process.

Synopsis: `#include <signal.h>`
`int raise(int sig);`

Derivation: First released in Issue 4. Derived from the ANSI C standard.

Issue 7: Functionality relating to the Threads option is moved to the Base.

rand, rand_r, srand

Purpose: Pseudo-random number generator.

Synopsis: `#include <stdlib.h>`
`int rand(void);`
OB CX `int rand_r(unsigned *seed);`
`void srand(unsigned seed);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The `rand_r()` function is marked obsolescent. Applications should use `random()` instead, or `erand48()`, `rand48()`, or `rand48()` when an independent random number sequence in multiple threads is required.

pread, read

Purpose: Read from a file.

Synopsis:

```
#include <unistd.h>

ssize_t pread(int fildes, void *buf, size_t nbyte,
              off_t offset);
ssize_t read(int fildes, void *buf, size_t nbyte);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The `pread()` function is moved from the XSI option to the Base.
 Functionality relating to the XSI STREAMS option is marked obsolescent.
 Changes are made related to support for finegrained timestamps.

readdir, readdir_r

Purpose: Read a directory.

Synopsis:

```
#include <dirent.h>

struct dirent *readdir(DIR *dirp);
int readdir_r(DIR *restrict dirp,
              struct dirent *restrict entry,
              struct dirent **restrict result);
```

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #059 is applied, updating the ERRORS section.
 The `readdir_r()` function is moved from the Thread-Safe Functions option to the Base.
 Changes are made related to support for finegrained timestamps.
 The value of the `d_ino` member is no longer unspecified for symbolic links.

readlink, readlinkat

Purpose: Read the contents of a symbolic link relative to a directory file descriptor.

Synopsis:

```
#include <unistd.h>

ssize_t readlink(const char *restrict path,
                 char *restrict buf, size_t bufsize);
ssize_t readlinkat(int fd, const char *restrict path,
                  char *restrict buf, size_t bufsize);
```

The `readlinkat()` function is equivalent to the `readlink()` function except in the case where `path` specifies a relative path. In this case the symbolic link whose content is read is relative to the directory associated with the file descriptor `fd` instead of the current working directory. If the file descriptor was opened without `O_SEARCH`, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with `O_SEARCH`, the function does not perform the check.

The purpose of the `readlinkat()` function is to read the content of symbolic links in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to

`readlink()`, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the `readlinkat()` function it can be guaranteed that the symbolic link read is located relative to the desired directory.

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than `{PATH_MAX}`.

The `readlinkat()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The `[EACCES]` error is removed from the “may fail” error conditions.

The `[ENOTDIR]` error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

readv

Purpose: Read a vector.

XSI Synopsis:

```
#include <sys/uio.h>
ssize_t readv(int fildes, const struct iovec *iov,
              int iovcnt);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Changes are made related to support for finegrained timestamps.

realloc

Purpose: Memory reallocator.

XSI Synopsis:

```
#include <stdlib.h>
void *realloc(void *ptr, size_t size);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

realpath

Purpose: Resolve a pathname.

XSI Synopsis:

```
#include <stdlib.h>
char *realpath(const char *restrict file_name,
               char *restrict resolved_name);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than `{PATH_MAX}`.

This function is updated for passing a null pointer for the `resolved_name` argument, to request that it allocate memory for the generated pathname, as if by `malloc()`. If `resolved_name` is not a null pointer and `{PATH_MAX}` is not defined as a constant in the `<limits.h>` header, the behavior is undefined.

recv

Purpose: Receive a message from a connected socket.

Synopsis:

```
#include <sys/socket.h>

ssize_t recv(int socket, void *buffer, size_t length,
             int flags);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

recvfrom

Purpose: Receive a message from a socket.

Synopsis:

```
#include <sys/socket.h>

ssize_t recvfrom(int socket, void *restrict buffer,
                 size_t length, int flags,
                 struct sockaddr *restrict address,
                 socklen_t *restrict address_len);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

recvmsg

Purpose: Receive a message from a socket.

Synopsis:

```
#include <sys/socket.h>

ssize_t recvmsg(int socket, struct msghdr *message, int flags);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

regcomp, regerror, regex, regfree

Purpose: Regular expression matching.

Synopsis:

```
#include <regex.h>

int regcomp(regex_t *restrict preg,
            const char *restrict pattern, int cflags);
size_t regerror(int errcode, const regex_t *restrict preg,
               char *restrict errbuf, size_t errbuf_size);
int regex(const regex_t *restrict preg,
          const char *restrict string, size_t nmatch,
          regmatch_t pmatch[restrict], int eflags);
void regfree(regex_t *preg);
```

Derivation: First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #134 is applied, clarifying that if more than one error occurs in processing a function call, any one of the possible constants may be returned.

SD5-XBD-ERN-60 is applied, removing the requirement that the type **regoff_t** can hold the largest value that can be stored in type **off_t**, and adding the requirement

that the type `regoff_t` can hold the largest value that can be stored in type `ptrdiff_t`.

remainder, remainderf, remainderl

Purpose: Remainder function.

Synopsis: `#include <math.h>`

```
double remainder(double x, double y);
float remainderf(float x, float y);
long double remainderl(long double x, long double y);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #55 (SD5-XSH-ERN-82) is applied.

remove

Purpose: Remove a file.

Synopsis: `#include <stdio.h>`

```
int remove(const char *path);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ISO C standard.

Issue 7: No functional changes are made in this issue.

remquo, remquof, remquol

Purpose: Remainder functions.

Synopsis: `#include <math.h>`

```
double remquo(double x, double y, int *quo);
float remquof(float x, float y, int *quo);
long double remquol(long double x, long double y, int *quo);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

Issue 7: ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #56 (SD5-XSH-ERN-83) is applied.

rename, renameat

Purpose: Rename file relative to directory file descriptor.

Synopsis: `#include <stdio.h>`

```
int rename(const char *old, const char *new);
int renameat(int oldfd, const char *old, int newfd,
             const char *new);
```

CX

The `renameat()` function is equivalent to the `rename()` function except in the case where either `old` or `new` specifies a relative path. If `old` is a relative path, the file to be renamed is located relative to the directory associated with the file descriptor `oldfd` instead of the current working directory. If `new` is a relative path, the same happens only relative to the directory associated with `newfd`. If the file descriptor

was opened without `O_SEARCH`, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with `O_SEARCH`, the function does not perform the check.

The purpose of the `renameat()` function is to rename files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to `rename()`, resulting in unspecified behavior. By opening file descriptors for the source and target directories and using the `renameat()` function it can be guaranteed that that renamed file is located correctly and the resulting file is in the desired directory.

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #016 is applied, changing the definition of the `[ENOTDIR]` error.

Austin Group Interpretation 1003.1-2001 #076 is applied, clarifying the behavior if the final component of a path is either dot or dot-dot, and adding the associated `[EINVAL]` error case.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than `{PATH_MAX}`.

Austin Group Interpretation 1003.1-2001 #145 is applied, clarifying that the `[ENOENT]` error condition also applies to the case in which a component of *new* does not exist.

Austin Group Interpretation 1003.1-2001 #181 is applied, updating the requirements for operations when the `S_ISVTX` bit is set on a directory.

The `renameat()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

rewind

Purpose: Reset the file position indicator in a stream.

Synopsis:

```
#include <stdio.h>

void rewind(FILE *stream);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

rewinddir

Purpose: Reset the position of a directory stream to the beginning of a directory.

Synopsis:

```
#include <dirent.h>

void rewinddir(DIR *dirp);
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

rint, rintf, rintl

Purpose: Round-to-nearest integral value.

Synopsis: `#include <math.h>`
`double rint(double x);`
`float rintf(float x);`
`long double rintl(long double x);`

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

rmdir

Purpose: Remove a directory.

Synopsis: `#include <unistd.h>`
`int rmdir(const char *path);`

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

Austin Group Interpretation 1003.1-2001 #181 is applied, updating the requirements for operations when the S_ISVTX bit is set.

Changes are made related to support for finegrained timestamps.

round, roundf, roundl

Purpose: Round to the nearest integer value in a floating-point format.

Synopsis: `#include <math.h>`
`double round(double x);`
`float roundf(float x);`
`long double roundl(long double x);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

scalbn, scalbnf, scalbnl, scalbn, scalbnf, scalbnl

Purpose: Compute exponent using FLT_RADIX.

Synopsis: `#include <math.h>`
`double scalbn(double x, long n);`
`float scalbnf(float x, long n);`
`long double scalbnl(long double x, long n);`
`double scalbn(double x, int n);`
`float scalbnf(float x, int n);`
`long double scalbnl(long double x, int n);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

sched_get_priority_max, sched_get_priority_min

Purpose: Get priority limits (**REALTIME**).

PS|TPS

Synopsis:

```
#include <sched.h>

int sched_get_priority_max(int policy);
int sched_get_priority_min(int policy);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: No functional changes are made in this issue.

sched_getparam

Purpose: Get scheduling parameters (**REALTIME**).

PS

Synopsis:

```
#include <sched.h>

int sched_getparam(pid_t pid, struct sched_param *param);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: No functional changes are made in this issue.

sched_getscheduler

Purpose: Get scheduling policy (**REALTIME**).

PS

Synopsis:

```
#include <sched.h>

int sched_getscheduler(pid_t pid);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: No functional changes are made in this issue.

sched_rr_get_interval

Purpose: Get execution time limits (**REALTIME**).

PS|TPS

Synopsis:

```
#include <sched.h>

int sched_rr_get_interval(pid_t pid,
    struct timespec *interval);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: No functional changes are made in this issue.

sched_setparam

Purpose: Set scheduling parameters (**REALTIME**).

PS

Synopsis:

```
#include <sched.h>

int sched_setparam(pid_t pid,
    const struct sched_param *param);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: Austin Group Interpretation 1003.1-2001 #061 is applied, clarifying the effect of process scheduling on the scheduling of threads within the process.

Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements for the *sched_ss_repl_period* and *sched_ss_init_budget* values.

sched_setscheduler

Purpose: Set scheduling policy and parameters (**REALTIME**).

PS

Synopsis:

```
#include <sched.h>

int sched_setscheduler(pid_t pid, int policy,
    const struct sched_param *param);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: Austin Group Interpretation 1003.1-2001 #061 is applied, clarifying the effect of process scheduling on the scheduling of threads within the process.

Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements for the *sched_ss_repl_period* and *sched_ss_init_budget* values.

sched_yield

Purpose: Yield the processor.

Synopsis:

```
#include <sched.h>

int sched_yield(void);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 7: SD5-XSH-ERN-120 is applied, adding APPLICATION USAGE.

The *sched_yield()* function is moved to the Base.

seekdir

Purpose: Set the position of a directory stream.

XSI

Synopsis:

```
#include <dirent.h>

void seekdir(DIR *dirp, long loc);
```


Derivation: First released in Issue 2.

Issue 7: SD5-XSH-ERN-200 is applied, updating the DESCRIPTION to note that the value of *loc* should have been returned from an earlier call to *telldir()* using the same directory stream.

sem_close

Purpose: Close a named semaphore.

Synopsis:

```
#include <semaphore.h>
int sem_close(sem_t *sem);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: The *sem_close()* function is moved from the Semaphores option to the Base.

sem_destroy

Purpose: Destroy an unnamed semaphore.

Synopsis:

```
#include <semaphore.h>
int sem_destroy(sem_t *sem);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: The *sem_destroy()* function is moved from the Semaphores option to the Base.

sem_getvalue

Purpose: Get the value of a semaphore.

Synopsis:

```
#include <semaphore.h>
int sem_getvalue(sem_t *restrict sem, int *restrict sval);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: The *sem_getvalue()* function is moved from the Semaphores option to the Base.

sem_init

Purpose: Initialize an unnamed semaphore.

Synopsis:

```
#include <semaphore.h>
int sem_init(sem_t *sem, int pshared, unsigned value);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: SD5-XSH-ERN-176 is applied.

The *sem_init()* function is moved from the Semaphores option to the Base.

sem_open

Purpose: Initialize and open a named semaphore.

Synopsis:

```
#include <semaphore.h>

sem_t *sem_open(const char *name, int oflag, ...);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: Austin Group Interpretation 1003.1-2001 #066 is applied, updating the [ENOSPC] error case and adding the [ENOMEM] error case.

Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and adding [ENAMETOOLONG] as a “may fail” error.

Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the user ID and group ID of the semaphore.

The *sem_open()* function is moved from the Semaphores option to the Base.

sem_post

Purpose: Unlock a semaphore.

Synopsis:

```
#include <semaphore.h>

int sem_post(sem_t *sem);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: The *sem_post()* function is moved from the Semaphores option to the Base.

sem_timedwait

Purpose: Lock a semaphore.

Synopsis:

```
#include <semaphore.h>
#include <time.h>

int sem_timedwait(sem_t *restrict sem,
                  const struct timespec *restrict abstime);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

Issue 7: The *sem_timedwait()* function is moved from the Semaphores option to the Base.

Functionality relating to the Timers option is moved to the Base.

An example is added.

sem_trywait, sem_wait

Purpose: Lock a semaphore.

Synopsis:

```
#include <semaphore.h>

int sem_trywait(sem_t *sem);
int sem_wait(sem_t *sem);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: SD5-XSH-ERN-54 is applied, removing the `sem_wait()` function from the “shall fail” error cases.

The `sem_trywait()` and `sem_wait()` functions are moved from the Semaphores option to the Base.

sem_unlink

Purpose: Remove a named semaphore.

Synopsis:

```
#include <semaphore.h>

int sem_unlink(const char *name);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error.

Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

The `sem_unlink()` function is moved from the Semaphores option to the Base.

semctl

Purpose: XSI semaphore control operations.

XSI Synopsis:

```
#include <sys/sem.h>

int semctl(int semid, int semnum, int cmd, ...);
```

Derivation: First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 7: No functional changes are made in this issue.

semget

Purpose: Get set of XSI semaphores.

XSI Synopsis:

```
#include <sys/sem.h>

int semget(key_t key, int nsems, int semflg);
```

Derivation: First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 7: No functional changes are made in this issue.

semop

Purpose: XSI semaphore operations.

XSI Synopsis:

```
#include <sys/sem.h>
int semop(int semid, struct sembuf *sops, size_t nsops);
```

Derivation: First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 7: SD5-XSH-ERN-171 is applied, updating the DESCRIPTION to clarify the order in which the operations in *sops* will be performed when there are multiple operations.

send

Purpose: Send a message on a socket.

Synopsis:

```
#include <sys/socket.h>
ssize_t send(int socket, const void *buffer,
             size_t length, int flags);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: Austin Group Interpretation 1003.1-2001 #035 is applied, updating the DESCRIPTION to clarify the behavior when the socket is a connectionless-mode socket.

The MSG_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The [EPIPE] error is modified.

sendmsg

Purpose: Send a message on a socket using a message structure.

Synopsis:

```
#include <sys/socket.h>
ssize_t sendmsg(int socket, const struct msghdr *message,
                int flags);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: Austin Group Interpretation 1003.1-2001 #073 is applied, describing the allowed behaviors when a peer address has been pre-specified.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The MSG_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The [EPIPE] error is modified.

sendto

Purpose: Send a message on a socket.

Synopsis:

```
#include <sys/socket.h>

ssize_t sendto(int socket, const void *message, size_t length,
               int flags, const struct sockaddr *dest_addr,
               socklen_t dest_len);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: Austin Group Interpretations 1003.1-2001 #035 and #073 are applied, describing the allowed behaviors when a peer address has been pre-specified.

Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The MSG_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The [EPIPE] error is modified.

setbuf

Purpose: Assign buffering to a stream.

Synopsis:

```
#include <stdio.h>

void setbuf(FILE *restrict stream, char *restrict buf);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

setegid

Purpose: Set the effective group ID.

Synopsis:

```
#include <unistd.h>

int setegid(gid_t gid);
```

Derivation: First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

Issue 7: No functional changes are made in this issue.

setenv

Purpose: Add or change environment variable.

cx Synopsis:

```
#include <stdlib.h>

int setenv(const char *envname, const char *envval,
           int overwrite);
```

Derivation: First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

Issue 7: No functional changes are made in this issue.

seteuid

Purpose: Set effective user ID.

Synopsis: `#include <unistd.h>`
`int seteuid(uid_t uid);`

Derivation: First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

Issue 7: No functional changes are made in this issue.

setgid

Purpose: Set group ID.

Synopsis: `#include <unistd.h>`
`int setgid(gid_t gid);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

setjmp

Purpose: Set jump point for a non-local goto.

Synopsis: `#include <setjmp.h>`
`int setjmp(jmp_buf env);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

setkey

Purpose: Set encoding key (**CRYPT**).

XSI Synopsis: `#include <stdlib.h>`
`void setkey(const char *key);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

setlocale

Purpose: Set program locale.

Synopsis: `#include <locale.h>`
`char *setlocale(int category, const char *locale);`

Derivation: First released in Issue 3.

Issue 7: Functionality relating to the Threads option is moved to the Base.

setpgid

Purpose: Set process group ID for job control.

Synopsis: `#include <unistd.h>`
`int setpgid(pid_t pid, pid_t pgid);`

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

setpgrp

Purpose: Set the process group ID.

OB XSI Synopsis: `#include <unistd.h>`
`pid_t setpgrp(void);`

Derivation: First released in Issue 4, Version 2.

Issue 7: The `setpgrp()` function is marked obsolescent. Applications should use `setpgid()` or `setsid()` as appropriate.

setregid

Purpose: Set real and effective group IDs.

XSI Synopsis: `#include <unistd.h>`
`int setregid(gid_t rgid, gid_t egid);`

Derivation: First released in Issue 4, Version 2.

Issue 7: SD5-XSH-ERN-177 is applied, adding the ability to set both the effective group ID and saved set-group-ID to be the same as the real group ID.

setreuid

Purpose: Set real and effective user IDs.

XSI Synopsis: `#include <unistd.h>`
`int setreuid(uid_t ruid, uid_t euid);`

Derivation: First released in Issue 4, Version 2.

Issue 7: SD5-XSH-ERN-177 is applied, adding the ability to set both the effective user ID and the saved set-user-ID to be the same as the real user ID.

setsid

Purpose: Create session and set process group ID.

Synopsis: `#include <unistd.h>`
`pid_t setsid(void);`

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

setsockopt

Purpose: Set the socket options.

Synopsis:

```
#include <sys/socket.h>

int setsockopt(int socket, int level, int option_name,
               const void *option_value, socklen_t option_len);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: Austin Group Interpretation 1003.1-2001 #158 is applied, removing text relating to socket options that is now in XSH Section 2.10.16 .

setuid

Purpose: Set user ID.

Synopsis:

```
#include <unistd.h>

int setuid(uid_t uid);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

setvbuf

Purpose: Assign buffering to a stream.

Synopsis:

```
#include <stdio.h>

int setvbuf(FILE *restrict stream, char *restrict buf,
            int type, size_t size);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

shm_open

Purpose: Open a shared memory object (**REALTIME**).

SHM Synopsis:

```
#include <sys/mman.h>

int shm_open(const char *name, int oflag, mode_t mode);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error.

Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the user ID and group ID of the shared memory object.

shm_unlink

Purpose: Remove a shared memory object (**REALTIME**).

SHM Synopsis:

```
#include <sys/mman.h>
int shm_unlink(const char *name);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error.
Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

shmat

Purpose: XSI shared memory attach operation.

XSI Synopsis:

```
#include <sys/shm.h>
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

Derivation: First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 7: No functional changes are made in this issue.

shmctl

Purpose: XSI shared memory control operations.

XSI Synopsis:

```
#include <sys/shm.h>
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

Derivation: First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 7: No functional changes are made in this issue.

shmdt

Purpose: XSI shared memory detach operation.

XSI Synopsis:

```
#include <sys/shm.h>
int shmdt(const void *shmaddr);
```

Derivation: First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 7: No functional changes are made in this issue.

shmget

Purpose: Get an XSI shared memory segment.

XSI Synopsis:

```
#include <sys/shm.h>
int shmget(key_t key, size_t size, int shmflg);
```

Derivation: First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 7: No functional changes are made in this issue.

shutdown

Purpose: Shut down socket send and receive operations.

Synopsis:

```
#include <sys/socket.h>
int shutdown(int socket, int how);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

sigaction

Purpose: Examine and change a signal action.

CX Synopsis:

```
#include <signal.h>
int sigaction(int sig, const struct sigaction *restrict act,
struct sigaction *restrict oact);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #004 is applied, clarifying that the *sigaction()* function may fail if the SA_SIGINFO flag is set in the *sa_flags* field of the **sigaction** structure for a signal not in the range SIGRTMIN to SIGRTMAX.

Austin Group Interpretations 1003.1-2001 #065 and #084 are applied, clarifying the role of the SA_NODEFER flag with respect to the signal mask, and clarifying the SA_RESTART flag for interrupted functions which use timeouts.

SD5-XSH-ERN-167 is applied, updating the APPLICATION USAGE section to explain that unless all signal handlers have *errno* set on return as it was on entry, the value of *errno* is unspecified.

SD5-XSH-ERN-172 is applied, updating the DESCRIPTION to make optional the requirement that when the SA_RESETHAND flag is set, *sigaction()* shall behave as if the SA_NODEFER flag were also set.

Functionality relating to the Realtime Signals Extension option is moved to the Base.

The description of the *si_code* member is replaced with a reference to XSH Section 2.4.3 .

sigaddset

Purpose: Add a signal to a signal set.

cx Synopsis:

```
#include <signal.h>
int sigaddset(sigset_t *set, int signo);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

sigaltstack

Purpose: Set and get signal alternate stack context.

xsi Synopsis:

```
#include <signal.h>
int sigaltstack(const stack_t *restrict ss,
stack_t *restrict oss);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

sigdelset

Purpose: Delete a signal from a signal set.

cx Synopsis:

```
#include <signal.h>
int sigdelset(sigset_t *set, int signo);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

sigemptyset

Purpose: Initialize and empty a signal set.

cx Synopsis:

```
#include <signal.h>
int sigemptyset(sigset_t *set);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

sigfillset

Purpose: Initialize and fill a signal set.

cx Synopsis:

```
#include <signal.h>
int sigfillset(sigset_t *set);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

sighold, sigignore, sigpause, sigrelse, sigset

Purpose: Signal management.

OB XSI Synopsis:

```
#include <signal.h>
int sighold(int sig);
int sigignore(int sig);
int sigpause(int sig);
int sigrelse(int sig);
void (*sigset(int sig, void (*disp)(int)))(int);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: These functions are marked obsolescent. Applications should use the *sigaction()* function instead of the *sigset()* function, the *pthread_sigmask()* or *sigprocmask()* functions instead of the *sighold()* and *sigrelse()* functions, and the *sigsuspend()* function instead of the *sigpause()* function.

siginterrupt

Purpose: Allow signals to interrupt functions.

OB XSI Synopsis:

```
#include <signal.h>
int siginterrupt(int sig, int flag);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The *siginterrupt()* function is marked obsolescent. Applications should use *sigaction()* with the SA_RESTART flag instead.

sigismember

Purpose: Test for a signal in a signal set.

CX Synopsis:

```
#include <signal.h>
int sigismember(const sigset_t *set, int signo);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

siglongjmp

Purpose: Non-local goto with signal handling.

CX Synopsis:

```
#include <setjmp.h>
void siglongjmp(sigjmp_buf env, int val);
```

Derivation: First released in Issue 3. Included for alignment with the ISO POSIX-1 standard.

Issue 7: No functional changes are made in this issue.

signal

Purpose: Signal management.

Synopsis: `#include <signal.h>`
`void (*signal(int sig, void (*func)(int)))(int);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

signbit

Purpose: Test sign.

Synopsis: `#include <math.h>`
`int signbit(real-floating x);`

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

sigpending

Purpose: Examine pending signals.

cx Synopsis: `#include <signal.h>`
`int sigpending(sigset_t *set);`

Derivation: First released in Issue 3.

Issue 7: No functional changes are made in this issue.

sigqueue

Purpose: Queue a signal to a process.

cx Synopsis: `#include <signal.h>`
`int sigqueue(pid_t pid, int signo, const union sigval value);`

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 7: The `sigqueue()` function is moved from the Realtime Signals Extension option to the Base.

sigsetjmp

Purpose: Set jump point for a non-local goto.

cx Synopsis: `#include <setjmp.h>`
`int sigsetjmp(sigjmp_buf env, int savemask);`

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

sigsuspend

Purpose: Wait for a signal.

cx Synopsis:

```
#include <signal.h>
int sigsuspend(const sigset_t *sigmask);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: SD5-XSH-ERN-122 is applied, adding the example code in the RATIONALE.

sigtimedwait, sigwaitinfo

Purpose: Wait for queued signals.

cx Synopsis:

```
#include <signal.h>
int sigtimedwait(const sigset_t *restrict set,
    siginfo_t *restrict info,
    const struct timespec *restrict timeout);
int sigwaitinfo(const sigset_t *restrict set,
    siginfo_t *restrict info);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 7: The *sigtimedwait()* and *sigwaitinfo()* functions are moved from the Realtime Signals Extension option to the Base.

sigwait

Purpose: Wait for queued signals.

cx Synopsis:

```
#include <signal.h>
int sigwait(const sigset_t *restrict set, int *restrict sig);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 7: Functionality relating to the Realtime Signals Extension option is moved to the Base.

sin, sinf, sinl

Purpose: Sine function.

Synopsis:

```
#include <math.h>
double sin(double x);
float sinf(float x);
long double sinl(long double x);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

sinh, sinhf, sinh1

Purpose: Hyperbolic sine functions.

Synopsis:

```
#include <math.h>

double sinh(double x);
float  sinhf(float x);
long double sinh1(long double x);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

sleep

Purpose: Suspend execution for an interval of time.

Synopsis:

```
#include <unistd.h>

unsigned sleep(unsigned seconds);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

socketmark

Purpose: Determine whether a socket is at the out-of-band mark.

Synopsis:

```
#include <sys/socket.h>

int socketmark(int s);
```

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

Issue 7: SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

socket

Purpose: Create an endpoint for communication.

Synopsis:

```
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

socketpair

Purpose: Create a pair of connected sockets.

Synopsis:

```
#include <sys/socket.h>

int socketpair(int domain, int type, int protocol,
               int socket_vector[2]);
```

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: The description of the [EMFILE] error condition is aligned with the *pipe()* function.

sqrt, sqrtf, sqrtl

Purpose: Square root function.

Synopsis:

```
#include <math.h>

double sqrt(double x);
float sqrtf(float x);
long double sqrtl(long double x);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

stderr, stdin, stdout

Purpose: Standard I/O streams.

Synopsis:

```
#include <stdio.h>

extern FILE *stderr, *stdin, *stdout;
```

Derivation: First released in Issue 1.

Issue 7: No functional changes are made in this issue.

strcasecmp, strcasecmp_l, strncasecmp, strncasecmp_l

Purpose: Case-insensitive string comparisons.

Synopsis:

```
#include <strings.h>

int strcasecmp(const char *s1, const char *s2);
int strcasecmp_l(const char *s1, const char *s2,
                 locale_t locale);
int strncasecmp(const char *s1, const char *s2, size_t n);
int strncasecmp_l(const char *s1, const char *s2,
                  size_t n, locale_t locale);
```

The *strcasecmp_l()* function compares, while ignoring differences in case, the string pointed to by *s1* to the string pointed to by *s2*. The *strncasecmp_l()* function compares, while ignoring differences in case, not more than *n* bytes from the string pointed to by *s1* to the string pointed to by *s2*.

These functions use the locale represented by *locale* to determine the case of the characters. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 4, Version 2.

Issue 7: The *strcasecmp()* and *strncasecmp()* functions are moved from the XSI option to the Base.

The *strcasecmp_l()* and *strncasecmp_l()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

strcat

Purpose: Concatenate two strings.

Synopsis: `#include <string.h>`
`char *strcat(char *restrict s1, const char *restrict s2);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

strchr

Purpose: String scanning operation.

Synopsis: `#include <string.h>`
`char *strchr(const char *s, int c);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

strcmp

Purpose: Compare two strings.

Synopsis: `#include <string.h>`
`int strcmp(const char *s1, const char *s2);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

strcoll, strcoll_l

Purpose: String comparison using collating information.

Synopsis: `#include <string.h>`
`int strcoll(const char *s1, const char *s2);`
`int strcoll_l(const char *s1, const char *s2,`
`locale_t locale);`

CX

The `strcoll_l()` function compares the string pointed to by `s1` to the string pointed to by `s2`, both interpreted as appropriate to the `LC_COLLATE` category of the locale represented by `locale`.

A handle for use as `locale` can be obtained using `newlocale()` or `duplocale()`.

Derivation: First released in Issue 3.

Issue 7: The `strcoll_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

stpcpy, strcpy

Purpose: Copy a string and return a pointer to the end of the result.

Synopsis: `#include <string.h>`

cx

```
char *stpcpy(char *restrict s1, const char *restrict s2);
char *strcpy(char *restrict s1, const char *restrict s2);
```

The *stpcpy()* function is equivalent to the *strcpy()* function, except that it returns a pointer to the terminating NUL character copied into the *s1* buffer.

The following example constructs a multi-part message in a single buffer:

```
#include <string.h>
#include <stdio.h>

int
main (void)
{
    char buffer [10];
    char *name = buffer;

    name = stpcpy (stpcpy (stpcpy (name, "ice"), "-"), "cream");
    puts (buffer);
    return 0;
}
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *stpcpy()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

strcspn

Purpose: Get the length of a complementary substring.

Synopsis: `#include <string.h>`

```
size_t strcspn(const char *s1, const char *s2);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

strdup, strndup

Purpose: Duplicate a specific number of bytes from a string.

cx

Synopsis: `#include <string.h>`

```
char *strdup(const char *s);
char *strndup(const char *s, size_t size);
```

The *strndup()* function is equivalent to the *strdup()* function, duplicating the provided *s* in a new block of memory allocated as if by using *malloc()*, with the exception being that *strndup()* copies at most *size* plus one bytes into the newly allocated memory, terminating the new string with a NUL character. If the length of *s* is larger than *size*, only *size* bytes are duplicated. If *size* is larger than the length of *s*, all bytes in *s* are copied into the new memory buffer, including the terminating NUL character. The newly created string is always properly terminated.

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [ENOMEM] error to become a “shall fail” error.

The *strdup()* function is moved from the XSI option to the Base.

The *strndup()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The APPLICATION USAGE section is updated to clarify that memory is allocated as if by *malloc()*.

strerror, strerror_l, strerror_r

Purpose: Get error message string.

Synopsis: `#include <string.h>`

```

char *strerror(int errnum);
CX char *strerror_l(int errnum, locale_t locale);
int strerror_r(int errnum, char *strerrbuf, size_t buflen);

```

Derivation: First released in Issue 3.

Issue 7: Austin Group Interpretation 1003.1-2001 #187 is applied, clarifying the behavior when the generated error message is an empty string.

SD5-XSH-ERN-191 is applied, disallowing *perror()* from overwriting the string returned by *strerror()*, for alignment with the C Standard.

The *strerror_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

The *strerror_r()* function is moved from the Thread-Safe Functions option to the Base.

strfmon, strfmon_l

Purpose: Convert monetary value to a string.

Synopsis: `#include <monetary.h>`

```

ssize_t strfmon(char *restrict s, size_t maxsize,
               const char *restrict format, ...);
ssize_t strfmon_l(char *restrict s, size_t maxsize,
                 locale_t locale, const char *restrict format, ...);

```

The *strfmon_l()* function is equivalent to the *strfmon()* function, except that the locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 4.

Issue 7: SD5-XSH-ERN-29 is applied, updating the examples for `%(#5n` and `%!(#5n`.

SD5-XSH-ERN-233 is applied, changing the definition of the `'+'` or `'('` flags to refer to multiple locales.

The *strfmon()* function is moved from the XSI option to the Base.

The *strfmon_I()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

strftime, strftime_l

Purpose: Convert date and time to a string.

Synopsis: `#include <time.h>`

```

size_t strftime(char *restrict s, size_t maxsize,
                const char *restrict format,
                const struct tm *restrict timeptr);
CX size_t strftime_l(char *restrict s, size_t maxsize,
                    const char *restrict format,
                    const struct tm *restrict timeptr, locale_t locale);

```

Derivation: First released in Issue 3.

Issue 7: Austin Group Interpretation 1003.1-2001 #163 is applied, making extensive changes to the required behavior of the *strftime()* function, including the addition of flags and field widths in conversion specifications.

The *strftime_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

strlen, strnlen

Purpose: Get length of fixed size string.

Synopsis: `#include <string.h>`

```

size_t strlen(const char *s);
CX size_t strnlen(const char *s, size_t maxlen);

```

The *strnlen()* function computes the smaller of the number of bytes in the array to which *s* points, not including the terminating NUL character, or the value of the *maxlen* argument. The *strnlen()* function never examines more than *maxlen* bytes of the array pointed to by *s*.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *strnlen()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

strncat

Purpose: Concatenate a string with part of another.

Synopsis: `#include <string.h>`

```

char *strncat(char *restrict s1, const char *restrict s2,
              size_t n);

```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

strncmp

Purpose: Compare part of two strings.

Synopsis: `#include <string.h>`

```
int strncmp(const char *s1, const char *s2, size_t n);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

stpncpy, strncpy

Purpose: Copy fixed length string, returning a pointer to the array end.

Synopsis: `#include <string.h>`

CX

```
char *stpncpy(char *restrict s1, const char *restrict s2,
              size_t n);
char *strncpy(char *restrict s1, const char *restrict s2,
              size_t n);
```

The *stpncpy()* function is equivalent to the *strncpy()* function, except for the return value. If a NUL character is written to the destination, the *stpncpy()* function returns the address of the first such NUL character. Otherwise, it returns `&s1[n]`.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *stpncpy()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

strpbrk

Purpose: Scan a string for a byte.

Synopsis: `#include <string.h>`

```
char *strpbrk(const char *s1, const char *s2);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

strptime

Purpose: Date and time conversion.

XSI

Synopsis: `#include <time.h>`

```
char *strptime(const char *restrict buf,
              const char *restrict format,
              struct tm *restrict tm);
```

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretations 1003.1-2001 #041 and #163 are applied, making extensive changes to the required behavior of the *strptime()* function, including the addition of flags and field widths in conversion specifications.

SD5-XSH-ERN-67 is applied, correcting the APPLICATION USAGE to remove the impression that `%Y` is 4-digit years.

strchr

Purpose: String scanning operation.

Synopsis: `#include <string.h>`
`char *strchr(const char *s, int c);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

strsignal

Purpose: Get name of signal.

cx Synopsis: `#include <string.h>`
`char *strsignal(int signum);`

The *strsignal()* function maps the signal number in *signum* to an implementation-defined string and returns a pointer to it. It uses the same set of messages as the *psignal()* function.

Application writers should note that if *signum* is not a valid signal number, some implementations return NULL, while for others the *strsignal()* function returns a pointer to a string containing an unspecified message denoting an unknown signal. POSIX.1-2008 leaves this return value unspecified.

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Issue 7: First released in Issue 7.

strspn

Purpose: Get length of a substring.

Synopsis: `#include <string.h>`
`size_t strspn(const char *s1, const char *s2);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

strstr

Purpose: Find a substring.

Synopsis: `#include <string.h>`
`char *strstr(const char *s1, const char *s2);`

Derivation: First released in Issue 3. Included for alignment with the ANSI C standard.

Issue 7: No functional changes are made in this issue.

strtod, strtodf, strtold

Purpose: Convert a string to a double-precision number.

Synopsis: #include <stdlib.h>

```
double strtod(const char *restrict nptr,
             char **restrict endptr);
float strtodf(const char *restrict nptr,
             char **restrict endptr);
long double strtold(const char *restrict nptr,
                  char **restrict endptr);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

strtoimax, strtoumax

Purpose: Convert string to integer type.

Synopsis: #include <inttypes.h>

```
intmax_t strtoimax(const char *restrict nptr,
                  char **restrict endptr, int base);
uintmax_t strtoumax(const char *restrict nptr,
                   char **restrict endptr, int base);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

strtok, strtok_r

Purpose: Split string into tokens.

Synopsis: #include <string.h>

CX

```
char *strtok(char *restrict s1, const char *restrict s2);
char *strtok_r(char *restrict s, const char *restrict sep,
              char **restrict lastsp);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: SD5-XSH-ERN-235 is applied, correcting an example.

The *strtok_r()* function is moved from the Thread-Safe Functions option to the Base.

strtol, strtoll

Purpose: Convert a string to a long integer.

Synopsis: #include <stdlib.h>

```
long strtol(const char *restrict str,
           char **restrict endptr, int base);
long long strtoll(const char *restrict str,
                 char **restrict endptr, int base);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

strtoul, strtoull

Purpose: Convert a string to an unsigned long.

Synopsis: `#include <stdlib.h>`

```
unsigned long strtoul(const char *restrict str,
                    char **restrict endptr, int base);
unsigned long long strtoull(const char *restrict str,
                           char **restrict endptr, int base);
```

Derivation: First released in Issue 4. Derived from the ANSI C standard.

Issue 7: No functional changes are made in this issue.

strxfrm, strxfrm_l

Purpose: String transformation.

Synopsis: `#include <string.h>`

```
size_t strxfrm(char *restrict s1, const char *restrict s2,
              size_t n);
CX size_t strxfrm_l(char *restrict s1, const char *restrict s2,
                 size_t n, locale_t locale);
```

The *strxfrm_l()* function is equivalent to the *strxfrm()* function, except that the locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 3. Included for alignment with the ISO C standard.

Issue 7: The *strxfrm_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

swab

Purpose: Swap bytes.

XSI

Synopsis: `#include <unistd.h>`

```
void swab(const void *restrict src, void *restrict dest,
         ssize_t nbytes);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

symlink, symlinkat

Purpose: Make a symbolic link relative to directory file descriptor.

Synopsis: `#include <unistd.h>`

```
int symlink(const char *path1, const char *path2);
int symlinkat(const char *path1, int fd, const char *path2);
```

The `symlinkat()` function is equivalent to the `symlink()` function except in the case where `path2` specifies a relative path. In this case the symbolic link is created relative to the directory associated with the file descriptor `fd` instead of the current working directory. If the file descriptor was opened without `O_SEARCH`, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with `O_SEARCH`, the function does not perform the check.

The purpose of the `symlinkat()` function is to create symbolic links in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to `symlink()`, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the `symlinkat()` function it can be guaranteed that the created symbolic link is located relative to the desired directory.

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than `{PATH_MAX}`.

The `symlinkat()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Additions have been made describing how `symlink()` sets the user and group IDs and file mode of the symbolic link, and its effect on timestamps.

sync

Purpose: Schedule file system updates.

XS1 Synopsis: `#include <unistd.h>`
`void sync(void);`

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

sysconf

Purpose: Get configurable system variables.

Synopsis: `#include <unistd.h>`

```
long sysconf(int name);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #160 is applied, clarifying the requirements related to variables that have no limit.

SD5-XSH-ERN-166 is applied, changing “Maximum size” to “Initial size” for the “Maximum size of ...” entries in the table in the DESCRIPTION.

The variables for the supported programming environments are updated to be V7 and the LEGACY variables are removed.

The following constants are added:

```
_POSIX_THREAD_ROBUST_PRIO_INHERIT
_POSIX_THREAD_ROBUST_PRIO_PROTECT
```

The `_XOPEN_UUCP` variable and its associated `_SC_XOPEN_UUCP` value is added to the table of system variables.

system

Purpose: Issue a command.

Synopsis: `#include <stdlib.h>`
`int system(const char *command);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #055 is applied, clarifying the thread-safety of this function and treatment of `pthread_atfork()` handlers.

tan, tanf, tanl

Purpose: Tangent function.

Synopsis: `#include <math.h>`
`double tan(double x);`
`float tanf(float x);`
`long double tanl(long double x);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

tanh, tanhf, tanhl

Purpose: Hyperbolic tangent functions.

Synopsis: `#include <math.h>`
`double tanh(double x);`
`float tanhf(float x);`
`long double tanhl(long double x);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

tcdrain

Purpose: Wait for transmission of output.

Synopsis: `#include <termios.h>`
`int tcdrain(int fildes);`

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

tcflow

Purpose: Suspend or restart the transmission or reception of data.

Synopsis:

```
#include <termios.h>
int tcflow(int fildev, int action);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: SD5-XSH-ERN-190 is applied, clarifying in the DESCRIPTION the transmission of START and STOP characters.

tcflush

Purpose: Flush non-transmitted output data, non-read input data, or both.

Synopsis:

```
#include <termios.h>
int tcflush(int fildev, int queue_selector);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

tcgetattr

Purpose: Get the parameters associated with the terminal.

Synopsis:

```
#include <termios.h>
int tcgetattr(int fildev, struct termios *termios_p);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

tcgetpgrp

Purpose: Get the foreground process group ID.

Synopsis:

```
#include <unistd.h>
pid_t tcgetpgrp(int fildev);
```

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

tcgetsid

Purpose: Get the process group ID for the session leader for the controlling terminal.

Synopsis:

```
#include <termios.h>
pid_t tcgetsid(int fildev);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: The *tcgetsid()* function is moved from the XSI option to the Base.

tcsendbreak

Purpose: Send a break for a specific duration.

Synopsis: `#include <termios.h>`
`int tcsendbreak(int fildes, int duration);`

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

tcsetattr

Purpose: Set the parameters associated with the terminal.

Synopsis: `#include <termios.h>`
`int tcsetattr(int fildes, int optional_actions,
const struct termios *termios_p);`

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #144 is applied, adding requirements related to the new `O_TTY_INIT` flag.

tcsetpgrp

Purpose: Set the foreground process group ID.

Synopsis: `#include <unistd.h>`
`int tcsetpgrp(int fildes, pid_t pgid_id);`

Derivation: First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 7: No functional changes are made in this issue.

tdelete, tfind, tsearch, twalk

Purpose: Manage a binary search tree.

XS1 Synopsis: `#include <search.h>`
`void *tdelete(const void *restrict key, void **restrict rootp,
int (*compar)(const void *, const void *));`
`void *tfind(const void *key, void *const *rootp,
int (*compar)(const void *, const void *));`
`void *tsearch(const void *key, void **rootp,
int (*compar)(const void *, const void *));`
`void twalk(const void *root,
void (*action)(const void *, VISIT, int));`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #149 is applied, clarifying concurrent use of the tree in another thread.

Austin Group Interpretation 1003.1-2001 #151 is applied, clarifying behavior for `tdelete()` when the deleted node is the root node.

Austin Group Interpretation 1003.1-2001 #153 is applied, clarifying that if the functions pointed to by `action` or `compar` change the tree, the results are

undefined.

telldir

Purpose: Current location of a named directory stream.

XSI Synopsis:

```
#include <dirent.h>
long telldir(DIR *dirp);
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

tempnam

Purpose: Create a name for a temporary file.

OB XSI Synopsis:

```
#include <stdio.h>
char *tempnam(const char *dir, const char *pfx);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *tempnam()* function is marked obsolescent. Applications should use the *tmpfile()*, *mkdtemp()*, or *mkstemp()* functions instead.

tgamma, tgammaf, tgammal

Purpose: Compute *gamma()* function.

Synopsis:

```
#include <math.h>
double tgamma(double x);
float tgammaf(float x);
long double tgammal(long double x);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #52 (SD5-XSH-ERN-85) is applied.

time

Purpose: Get time.

Synopsis:

```
#include <time.h>
time_t time(time_t *tloc);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

timer_create

Purpose: Create a per-process timer.

cx Synopsis:

```
#include <signal.h>
#include <time.h>

int timer_create(clockid_t clockid,
                 struct sigevent *restrict evp,
                 timer_t *restrict timerid);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: The *timer_create()* function is moved from the Timers option to the Base.

timer_delete

Purpose: Delete a per-process timer.

cx Synopsis:

```
#include <time.h>

int timer_delete(timer_t timerid);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: The *timer_delete()* function is moved from the Timers option to the Base.

timer_getoverrun, timer_gettime, timer_settime

Purpose: Per-process timers.

cx Synopsis:

```
#include <time.h>

int timer_getoverrun(timer_t timerid);
int timer_gettime(timer_t timerid, struct itimerspec *value);
int timer_settime(timer_t timerid, int flags,
                  const struct itimerspec *restrict value,
                  struct itimerspec *restrict ovalue);
```

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: The *timer_getoverrun()*, *timer_gettime()*, and *timer_settime()* functions are moved from the Timers option to the Base.

Functionality relating to the Realtime Signals Extension option is moved to the Base.

times

Purpose: Get process and waited-for child process times.

Synopsis:

```
#include <sys/times.h>
clock_t times(struct tms *buffer);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

tmpfile

Purpose: Create a temporary file.

Synopsis:

```
#include <stdio.h>
FILE *tmpfile(void);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #025 is applied, clarifying that implementations may restrict the permissions of the file created.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

SD5-XSH-ERN-149 is applied, adding the mandatory [EMFILE] error condition for {STREAM_MAX} streams open.

tmpnam

Purpose: Create a name for a temporary file.

OB Synopsis:

```
#include <stdio.h>
char *tmpnam(char *s);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *tmpnam()* function need not be thread-safe if called with a NULL parameter.

The *tmpnam()* function is marked obsolescent. Applications should use the *tmpfile()*, *mkdtemp()*, or *mkstemp()* functions instead.

toascii

Purpose: Translate an integer to a 7-bit ASCII character.

OB XSI Synopsis:

```
#include <ctype.h>
int toascii(int c);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *toascii()* function is marked obsolescent.

tolower, tolower_l

Purpose: Transliterate uppercase characters to lowercase.

Synopsis: `#include <ctype.h>`

```
int tolower(int c);
CX int tolower_l(int c, locale_t locale);
```

The *tolower_l()* function is equivalent to the *tolower()* function, except that the locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *tolower_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

toupper, toupper_l

Purpose: Transliterate lowercase characters to uppercase.

Synopsis: `#include <ctype.h>`

```
int toupper(int c);
CX int toupper_l(int c, locale_t locale);
```

The *toupper_l()* function is equivalent to the *toupper()* function, except that the locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *toupper_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

towctrans, towctrans_l

Purpose: Wide-character transliteration.

Synopsis: `#include <wctype.h>`

```
wint_t towctrans(wint_t wc, wctrans_t desc);
CX wint_t towctrans_l(wint_t wc, wctrans_t desc,
    locale_t locale);
```

The *towctrans_l()* function is equivalent to the *towctrans()* function, except that the locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: The *towctrans_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

towlower, towlower_l

Purpose: Transliterate uppercase wide-character code to lowercase.

Synopsis: `#include <wctype.h>`

```

wint_t towlower(wint_t wc);
CX  wint_t towlower_l(wint_t wc, locale_t locale);

```

The *towlower_l()* function is equivalent to the *towlower()* function, except that the locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 4.

Issue 7: The *towlower_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

towupper, towupper_l

Purpose: Transliterate lowercase wide-character code to uppercase.

Synopsis: `#include <wctype.h>`

```

wint_t towupper(wint_t wc);
CX  wint_t towupper_l(wint_t wc, locale_t locale);

```

The *towupper_l()* function is equivalent to the *towupper()* function, except that the locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 4.

Issue 7: The *towupper_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

trunc, truncf, truncf_l

Purpose: Round to truncated integer value.

Synopsis: `#include <math.h>`

```

double trunc(double x);
float truncf(float x);
long double truncf_l(long double x);

```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

truncate

Purpose: Truncate a file to a specified length.

Synopsis: `#include <unistd.h>`

```

int truncate(const char *path, off_t length);

```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The `truncate()` function is moved from the XSI option to the Base.

Changes are made related to support for finegrained timestamps.

The [ENOTDIR] error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

ttyname, ttyname_r

Purpose: Find the pathname of a terminal.

Synopsis: `#include <unistd.h>`

```
char *ttyname(int fildes);
int ttyname_r(int fildes, char *name, size_t namesize);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

The `ttyname_r()` function is moved from the Thread-Safe Functions option to the Base.

daylight, timezone, tzname, tzset

Purpose: Set timezone conversion information.

Synopsis: `#include <time.h>`

```
XSI      extern int daylight;
CX      extern long timezone;
        extern char *tzname[2];
        void tzset(void);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

ulimit

Purpose: Get and set process limits.

```
OB XSI  Synopsis: #include <ulimit.h>
        long ulimit(int cmd, ...);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The `ulimit()` function is marked obsolescent. Applications should use the `getrlimit()` or `setrlimit()` functions instead.

umask

Purpose: Set and get the file mode creation mask.

Synopsis: `#include <sys/stat.h>`
`mode_t umask(mode_t cmask);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

uname

Purpose: Get the name of the current system.

Synopsis: `#include <sys/utsname.h>`
`int uname(struct utsname *name);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

ungetc

Purpose: Push byte back into input stream.

Synopsis: `#include <stdio.h>`
`int ungetc(int c, FILE *stream);`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

ungetwc

Purpose: Push wide-character code back into the input stream.

Synopsis: `#include <stdio.h>`
`#include <wchar.h>`
`wint_t ungetwc(wint_t wc, FILE *stream);`

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: No functional changes are made in this issue.

unlink, unlinkat

Purpose: Remove a directory entry relative to directory file descriptor.

Synopsis: `#include <unistd.h>`
`int unlink(const char *path);`
`int unlinkat(int fd, const char *path, int flag);`

The `unlinkat()` function is equivalent to the `unlink()` or `rmdir()` function except in the case where `path` specifies a relative path. In this case the directory entry to be removed is determined relative to the directory associated with the file descriptor `fd` instead of the current working directory. If the file descriptor was opened without `O_SEARCH`, the function checks whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with `O_SEARCH`, the function does not perform the check.

The `AT_REMOVEDIR` flag controls whether `unlinkat()` behaves like `unlink()` or `rmdir()`: if `AT_REMOVEDIR` is set, the directory entry specified by `fd` and `path` is removed as a directory.

The purpose of the `unlinkat()` function is to remove directory entries in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to `unlink()`, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the `unlinkat()` function it can be guaranteed that the removed directory entry is located relative to the desired directory.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than `{PATH_MAX}`.

Austin Group Interpretation 1003.1-2001 #181 is applied, updating the requirements for operations when the `S_ISVTX` bit is set on a directory.

Text arising from the LSB Conflicts TR is added to the RATIONALE about the use of `[EPERM]` and `[EISDIR]`.

The `unlinkat()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

The `[ENOTDIR]` error condition is clarified to cover the condition where the last component of a pathname exists but is not a directory or a symbolic link to a directory.

unlockpt

Purpose: Unlock a pseudo-terminal master/slave pair.

xSI Synopsis:

```
#include <stdlib.h>
int unlockpt(int fildes);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

unsetenv

Purpose: Remove an environment variable.

cx Synopsis:

```
#include <stdlib.h>
int unsetenv(const char *name);
```

Derivation: First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

Issue 7: No functional changes are made in this issue.

uselocale

Purpose: Use locale in current thread.

CX Synopsis:

```
#include <locale.h>
locale_t uselocale(locale_t newloc);
```

The *uselocale()* function sets the current locale for the current thread to the locale represented by *newloc*.

Application writers should note that unlike the *setlocale()* function, the *uselocale()* function does not allow replacing some locale categories only. Applications that need to install a locale which differs only in a few categories must use *newlocale()* to change a locale object equivalent to the currently used locale and install it.

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 4.

Issue 7: First released in Issue 7.

utime

Purpose: Set file access and modification times.

OB Synopsis:

```
#include <utime.h>
int utime(const char *path, const struct utimbuf *times);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than {PATH_MAX}.

The *utime()* function is marked obsolescent. Applications should use the *utimensat()* function instead.

Changes are made related to support for finegrained timestamps.

vdprintf, vfprintf, vprintf, vsnprintf, vsprintf

Purpose: Format output of a stdarg argument list.

CX Synopsis:

```
#include <stdarg.h>
#include <stdio.h>
int vdprintf(int fildes, const char *restrict format,
va_list ap);
int vfprintf(FILE *restrict stream,
const char *restrict format, va_list ap);
int vprintf(const char *restrict format, va_list ap);
int vsnprintf(char *restrict s, size_t n,
const char *restrict format, va_list ap);
int vsprintf(char *restrict s, const char *restrict format,
va_list ap);
```

The *vdprintf()* function is equivalent to the *vfprintf()* function, except that *vdprintf()* writes output to the file associated with the file descriptor specified by the *fildes* argument rather than placing output on a stream.

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The `vdprintf()` function is added to complement the `dprintf()` function from The Open Group Technical Standard, 2006, Extended API Set Part 1.

vfscanf, vscanf, vsscanf

Purpose: Format input of a stdarg argument list.

Synopsis:

```
#include <stdarg.h>
#include <stdio.h>

int vfscanf(FILE *restrict stream, const char *restrict format,
            va_list arg);
int vscanf(const char *restrict format, va_list arg);
int vsscanf(const char *restrict s, const char *restrict format,
            va_list arg);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

vwprintf, vswprintf, wprintf

Purpose: Wide-character formatted output of a stdarg argument list.

Synopsis:

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vwprintf(FILE *restrict stream,
             const wchar_t *restrict format, va_list arg);
int vswprintf(wchar_t *restrict ws, size_t n,
             const wchar_t *restrict format, va_list arg);
int wprintf(const wchar_t *restrict format, va_list arg);
```

Derivation: First released in Issue 5. Included for alignment with ISO/IEC 9899: 1990/Amendment 1: 1995 (E).

Issue 7: No functional changes are made in this issue.

vwscanf, vswscanf, wscanf

Purpose: Wide-character formatted input of a stdarg argument list.

Synopsis:

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vwscanf(FILE *restrict stream,
            const wchar_t *restrict format, va_list arg);
int vswscanf(const wchar_t *restrict ws,
            const wchar_t *restrict format, va_list arg);
int wscanf(const wchar_t *restrict format, va_list arg);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

wait, waitpid

Purpose: Wait for a child process to stop or terminate.

Synopsis: `#include <sys/wait.h>`

```
pid_t wait(int *stat_loc);
pid_t waitpid(pid_t pid, int *stat_loc, int options);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: APPLICATION USAGE is added, recommending that the `wait()` function not be used and that the `waitpid()` function not be used with a `pid` argument of `-1`.

An additional example for `waitpid()` is added.

waitid

Purpose: Wait for a child process to change state.

Synopsis: `#include <sys/wait.h>`

```
int waitid(idtype_t idtype, id_t id, siginfo_t *infop,
           int options);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #060 is applied, updating the DESCRIPTION to require that applications set at least one of the flags WEXITED, WSTOPPED or WCONTINUED in the `options` argument.

The `waitid()` function is moved from the XSI option to the Base.

APPLICATION USAGE is added, recommending that the `waitid()` function not be used with `idtype` equal to `P_ALL`.

The description of the `WNOHANG` flag is updated to match the one on the `<sys/wait.h>` page.

wcrtomb

Purpose: Convert a wide-character code to a character (restartable).

Synopsis: `#include <stdio.h>`

```
size_t wcrtomb(char *restrict s, wchar_t wc,
               mbstate_t *restrict ps);
```

Derivation: First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the `wcrtomb()` function need not be thread-safe if called with a `NULL` `ps` argument.

Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a “may fail” to a “shall fail”.

wcscasecmp, wcscasecmp_l, wcsncasecmp, wcsncasecmp_l

Purpose: Case-insensitive wide-character string comparison.

cx Synopsis:

```
#include <wchar.h>

int wcscasecmp(const wchar_t *ws1, const wchar_t *ws2);
int wcscasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
    locale_t locale);
int wcsncasecmp(const wchar_t *ws1, const wchar_t *ws2,
    size_t n);
int wcsncasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
    size_t n, locale_t locale);
```

The *wcscasecmp()* and *wcsncasecmp()* functions are the wide-character equivalent of the *strcasecmp()* and *strncasecmp()* functions, respectively.

The *wcscasecmp()* and *wcscasecmp_l()* functions compare, while ignoring differences in case, the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*.

The *wcsncasecmp()* and *wcsncasecmp_l()* functions compare, while ignoring differences in case, not more than *n* wide-characters from the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*.

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Issue 7: First released in Issue 7.

wcscat

Purpose: Concatenate two wide-character strings.

Synopsis:

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *restrict ws1,
    const wchar_t *restrict ws2);
```

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: No functional changes are made in this issue.

wcschr

Purpose: Wide-character string scanning operation.

Synopsis:

```
#include <wchar.h>

wchar_t *wcschr(const wchar_t *ws, wchar_t wc);
```

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: No functional changes are made in this issue.

wcscmp

Purpose: Compare two wide-character strings.

Synopsis: `#include <wchar.h>`

```
int wcscmp(const wchar_t *ws1, const wchar_t *ws2);
```

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: No functional changes are made in this issue.

wcscoll, wcscoll_l

Purpose: Wide-character string comparison using collating information.

Synopsis: `#include <wchar.h>`

```
CX int wcscoll(const wchar_t *ws1, const wchar_t *ws2);
int wcscoll_l(const wchar_t *ws1, const wchar_t *ws2,
             locale_t locale);
```

The `wcscoll_l()` function compares the wide-character string pointed to by `ws1` to the wide-character string pointed to by `ws2`, both interpreted as appropriate to the `LC_COLLATE` category of the locale represented by `locale`.

A handle for use as `locale` can be obtained using `newlocale()` or `duplocale()`.

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: The `wcscoll_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

wcpcpy, wcscopy

Purpose: Copy a wide-character string, returning a pointer to its end.

Synopsis: `#include <wchar.h>`

```
CX wchar_t *wcpcpy(wchar_t *restrict ws1,
                const wchar_t *restrict ws2);
wchar_t *wcscopy(wchar_t *restrict ws1,
                const wchar_t *restrict ws2);
```

The `wcpcpy()` function is equivalent to the `wcscopy()` function, except that it returns a pointer to the terminating null wide-character code copied into the `ws1` buffer.

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: The `wcpcpy()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

wcscspn

Purpose: Get the length of a complementary wide substring.

Synopsis: `#include <wchar.h>`

```
size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2);
```

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: No functional changes are made in this issue.

wcsdup

Purpose: Duplicate a wide-character string.

cx Synopsis:

```
#include <wchar.h>
wchar_t *wcsdup(const wchar_t *string);
```

The *wcsdup()* function is the wide-character equivalent of the *strdup()* function.

Application writers should note that for functions that allocate memory as if by *malloc()*, (such as *wcsdup()*) the application should release such memory when it is no longer required by a call to *free()*. For *wcsdup()*, this is the return value.

Derivation: First released in Issue 7. Derived from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Issue 7: First released in Issue 7.

wcsftime

Purpose: Convert date and time to a wide-character string.

Synopsis:

```
#include <wchar.h>

size_t wcsftime(wchar_t *restrict wcs, size_t maxsize,
               const wchar_t *restrict format,
               const struct tm *restrict timeptr);
```

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

wcslen, wcsnlen

Purpose: Get length of a fixed-sized wide-character string.

cx Synopsis:

```
#include <wchar.h>

size_t wcslen(const wchar_t *ws);
size_t wcsnlen(const wchar_t *ws, size_t maxlen);
```

The *wcsnlen()* function computes the smaller of the number of wide characters in the string to which *ws* points, not including the terminating null wide-character code, and the value of *maxlen*. The *wcsnlen()* function never examines more than the first *maxlen* characters of the wide-character string pointed to by *ws*.

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: The *wcsnlen()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

wcsncat

Purpose: Concatenate a wide-character string with part of another.

Synopsis:

```
#include <wchar.h>

wchar_t *wcsncat(wchar_t *restrict ws1,
                 const wchar_t *restrict ws2, size_t n);
```

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: No functional changes are made in this issue.

wcsncmp

Purpose: Compare part of two wide-character strings.

Synopsis:

```
#include <wchar.h>

int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
```

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: No functional changes are made in this issue.

wcpncpy, wcsncpy

Purpose: Copy a fixed-size wide-character string, returning a pointer to its end.

Synopsis:

```
#include <wchar.h>

wchar_t *wcpncpy(wchar_t restrict *ws1,
                 const wchar_t *restrict ws2, size_t n);
wchar_t *wcsncpy(wchar_t *restrict ws1,
                 const wchar_t *restrict ws2, size_t n);
```

CX

The *wcpncpy()* function is equivalent to the *wcsncpy()* function, except for the return value. If any null wide-character codes were written into the destination, the *wcpncpy()* function returns the address of the first such null wide-character code. Otherwise, it returns *&ws1[n]*.

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: The *wcpncpy()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

wcspbrk

Purpose: Scan a wide-character string for a wide-character code.

Synopsis:

```
#include <wchar.h>

wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);
```

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: No functional changes are made in this issue.

wcsrchr

Purpose: Wide-character string scanning operation.

Synopsis: `#include <wchar.h>`
`wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);`

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: No functional changes are made in this issue.

wcsnrtombs, wcsrtombs

Purpose: Convert a wide-character string to a character string (restartable).

Synopsis: `#include <wchar.h>`

CX

```
size_t wcsnrtombs(char *restrict dst,
                  const wchar_t **restrict src, size_t nwc,
                  size_t len, mbstate_t *restrict ps);
size_t wcsrtombs(char *restrict dst,
                  const wchar_t **restrict src, size_t len,
                  mbstate_t *restrict ps);
```

The `wcsnrtombs()` function is equivalent to the `wcsrtombs()` function, except that the conversion is limited to the first `nwc` wide characters.

Derivation: First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the `wcsrtombs()` function need not be thread-safe if called with a NULL `ps` argument.

Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a “may fail” to a “shall fail”.

The `wcsnrtombs()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

wcsspn

Purpose: Get the length of a wide substring.

Synopsis: `#include <wchar.h>`
`size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);`

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: No functional changes are made in this issue.

wcsstr

Purpose: Find a wide-character substring.

Synopsis: `#include <wchar.h>`
`wchar_t *wcsstr(const wchar_t *restrict ws1,
 const wchar_t *restrict ws2);`

Derivation: First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: No functional changes are made in this issue.

wcstod, wcstof, wcstold

Purpose: Convert a wide-character string to a double-precision number.

Synopsis:

```
#include <wchar.h>

double wcstod(const wchar_t *restrict nptr,
              wchar_t **restrict endptr);
float wcstof(const wchar_t *restrict nptr,
             wchar_t **restrict endptr);
long double wcstold(const wchar_t *restrict nptr,
                   wchar_t **restrict endptr);
```

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: No functional changes are made in this issue.

wcstoimax, wcstoumax

Purpose: Convert a wide-character string to an integer type.

Synopsis:

```
#include <stddef.h>
#include <inttypes.h>

intmax_t wcstoimax(const wchar_t *restrict nptr,
                  wchar_t **restrict endptr, int base);
uintmax_t wcstoumax(const wchar_t *restrict nptr,
                   wchar_t **restrict endptr, int base);
```

Derivation: First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

Issue 7: No functional changes are made in this issue.

wcstok

Purpose: Split a wide-character string into tokens.

Synopsis:

```
#include <wchar.h>

wchar_t *wcstok(wchar_t *restrict ws1,
               const wchar_t *restrict ws2,
               wchar_t **restrict ptr);
```

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

wcstol, wcstoll

Purpose: Convert a wide-character string to a long integer.

Synopsis:

```
#include <wchar.h>

long wcstol(const wchar_t *restrict nptr,
            wchar_t **restrict endptr, int base);
long long wcstoll(const wchar_t *restrict nptr,
                 wchar_t **restrict endptr, int base);
```

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: No functional changes are made in this issue.

wcstombs

Purpose: Convert a wide-character string to a character string.

Synopsis: `#include <stdlib.h>`

```
size_t wcstombs(char *restrict s,
                const wchar_t *restrict pwcs, size_t n);
```

Derivation: First released in Issue 4. Derived from the ISO C standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #170 is applied, changing the [EILSEQ] error condition from a “may fail” to a “shall fail”.

wcstoul, wcstoull

Purpose: Convert a wide-character string to an unsigned long.

Synopsis: `#include <wchar.h>`

```
unsigned long wcstoul(const wchar_t *restrict nptr,
                    wchar_t **restrict endptr, int base);
unsigned long long wcstoull(const wchar_t *restrict nptr,
                           wchar_t **restrict endptr, int base);
```

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: No functional changes are made in this issue.

wcswidth

Purpose: Number of column positions of a wide-character string.

XSI Synopsis: `#include <wchar.h>`

```
int wcswidth(const wchar_t *pwcs, size_t n);
```

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: No functional changes are made in this issue.

wcsxfrm, wcsxfrm_l

Purpose: Wide-character string transformation.

Synopsis: `#include <wchar.h>`

```
size_t wcsxfrm(wchar_t *restrict ws1,
               const wchar_t *restrict ws2, size_t n);
CX size_t wcsxfrm_l(wchar_t *restrict ws1,
                  const wchar_t *restrict ws2, size_t n,
                  locale_t locale);
```

The `wcsxfrm_l()` function is equivalent to the `wcsxfrm()` function, except that the locale data used is from the locale represented by `locale`. A handle for use as `locale` can be obtained using `newlocale()` or `duplocale()`.

Derivation: First released in Issue 4. Derived from the MSE working draft.

Issue 7: The *wcsxfrm_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

wctob

Purpose: Wide-character to single-byte conversion.

Synopsis:

```
#include <stdio.h>
#include <wchar.h>

int wctob(wint_t c);
```

Derivation: First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: No functional changes are made in this issue.

wctomb

Purpose: Convert a wide-character code to a character.

Synopsis:

```
#include <stdlib.h>

int wctomb(char *s, wchar_t wchar);
```

Derivation: First released in Issue 4. Derived from the ANSI C standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #170 is applied, adding the [EILSEQ] error condition.

wctrans, wctrans_l

Purpose: Define character mapping.

Synopsis:

```
#include <wctype.h>

wctrans_t wctrans(const char *charclass);
wctrans_t wctrans_l(const char *charclass, locale_t locale);
```

CX

The *wctrans_l()* function is equivalent to the *wctrans()* function, except that the locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: The *wctrans_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

wctype, wctype_l

Purpose: Define character class.

Synopsis:

```
#include <wctype.h>

wctype_t wctype(const char *property);
wctype_t wctype_l(const char *property, locale_t locale);
```

CX

The *wctype_l()* function is equivalent to the *wctype()* function, except that the

locale data used is from the locale represented by *locale*. A handle for use as *locale* can be obtained using *newlocale()* or *duplocale()*.

Derivation: First released in Issue 4.

Issue 7: The *wctype_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

wcwidth

Purpose: Number of column positions of a wide-character code.

XSI Synopsis:

```
#include <wchar.h>
int wcwidth(wchar_t wc);
```

Derivation: First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working draft.

Issue 7: No functional changes are made in this issue.

wmemchr

Purpose: Find a wide character in memory.

Synopsis:

```
#include <wchar.h>
wchar_t *wmemchr(const wchar_t *ws, wchar_t wc, size_t n);
```

Derivation: First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: No functional changes are made in this issue.

wmemcmp

Purpose: Compare wide characters in memory.

Synopsis:

```
#include <wchar.h>
int wmemcmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
```

Derivation: First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: No functional changes are made in this issue.

wmemcpy

Purpose: Copy wide characters in memory.

Synopsis:

```
#include <wchar.h>
wchar_t *wmemcpy(wchar_t *restrict ws1,
const wchar_t *restrict ws2, size_t n);
```

Derivation: First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: No functional changes are made in this issue.

wmemmove

Purpose: Copy wide characters in memory with overlapping areas.

Synopsis: `#include <wchar.h>`

```
wchar_t *wmemmove(wchar_t *ws1, const wchar_t *ws2, size_t n);
```

Derivation: First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: No functional changes are made in this issue.

wmemset

Purpose: Set wide characters in memory.

Synopsis: `#include <wchar.h>`

```
wchar_t *wmemset(wchar_t *ws, wchar_t wc, size_t n);
```

Derivation: First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: No functional changes are made in this issue.

wordexp, wordfree

Purpose: Perform word expansions.

Synopsis: `#include <wordexp.h>`

```
int wordexp(const char *restrict words,
            wordexp_t *restrict pwordexp, int flags);
void wordfree(wordexp_t *pwordexp);
```

Derivation: First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #148 is applied, adding APPLICATION USAGE explaining that the `wordexp()` function need not be thread safe if passed an expression referencing an environment variable while any other thread is concurrently modifying any environment variable.

pwrite, write

Purpose: Write on a file.

Synopsis: `#include <unistd.h>`

```
ssize_t pwrite(int fildes, const void *buf, size_t nbyte,
               off_t offset);
ssize_t write(int fildes, const void *buf, size_t nbyte);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The `pwrite()` function is moved from the XSI option to the Base.

Functionality relating to the XSI STREAMS option is marked obsolescent.

SD5-XSH-ERN-160 is applied, updating the DESCRIPTION to clarify the requirements for the `pwrite()` function, and to change the use of the phrase “file pointer” to “file offset”.

writev

Purpose: Write a vector.

XSI Synopsis:

```
#include <sys/uio.h>
ssize_t writev(int fildes, const struct iovec *iov,
               int iovcnt);
```

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

y0, y1, yn

Purpose: Bessel functions of the second kind.

XSI Synopsis:

```
#include <math.h>
double y0(double x);
double y1(double x);
double yn(int n, double x);
```

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

Utilities Migration

12.1 Introduction

This chapter contains a section for each utility interface defined in XCU, Issue 7. Each section contains the SYNOPSIS and gives the derivation of the interface. Where new option letters have been added in Issue 7, a brief description is included, complete with examples where appropriate. For interfaces carried forward from Issue 6, syntax and semantic changes made to the interface in Issue 7 are identified (if any). Only changes that might affect an application programmer are included.

12.2 Utilities

admin

Purpose: Create and administer SCCS files (**DEVELOPMENT**).

XSI Synopsis:

```
admin -i[name] [-n] [-a login] [-d flag] [-e login] [-f flag]
      [-m mrlist] [-r rel] [-t[name] [-y[comment]]] newfile
admin -n [-a login] [-d flag] [-e login] [-f flag] [-m mrlist]
      [-t[name]] [-y[comment]] newfile...
admin [-a login] [-d flag] [-m mrlist] [-r rel]
      [-t[name]] file...
admin -h file...
admin -z file...
```

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

alias

Purpose: Define or display aliases.

Synopsis: `alias [alias-name[=string]...]`

Derivation: First released in Issue 4.

Issue 7: The *alias* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

The first example is changed to remove the creation of an alias for a standard utility that alters its behavior to be non-conforming.

ar

Purpose: Create and maintain library archives.

SD Synopsis: `ar -d [-v] archive file...`

XSI `ar -m [-v] archive file...`
`ar -m -a [-v] posname archive file...`
`ar -m -b [-v] posname archive file...`
`ar -m -i [-v] posname archive file...`

XSI `ar -p [-v] [-s] archive [file...]`

XSI `ar -q [-cv] archive file...`

`ar -r [-cuv] archive file...`

XSI `ar -r -a [-cuv] posname archive file...`
`ar -r -b [-cuv] posname archive file...`
`ar -r -i [-cuv] posname archive file...`

XSI `ar -t [-v] [-s] archive [file...]`

XSI `ar -x [-v] [-sCT] archive [file...]`

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply (options can be interspersed with operands).

asa

Purpose: Interpret carriage-control characters.

FR Synopsis: `asa [file...]`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is '-' and the implementation treats the '-' as meaning standard input.

at

Purpose: Execute commands at a later time.

Synopsis: `at [-m] [-f file] [-q queue_name] -t time_arg`
`at [-m] [-f file] [-q queue_name] timespec...`
`at -r at_job_id...`
`at -l -q queue_name`
`at -l [at_job_id...]`

Derivation: First released in Issue 2.

Issue 7: The *at* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced by the *at* utility.

awk

Purpose: Pattern scanning and processing language.

Synopsis: `awk [-F ERE] [-v assignment]... program [argument...]`
`awk [-F ERE] -f progfile [-f progfile]... [-v assignment]... [argument...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #189 is applied, changing the EXTENDED DESCRIPTION to make the support of hexadecimal integer and floating constants optional.

Austin Group Interpretation 1003.1-2001 #201 is applied, permitting implementations to support infinities and NaNs.

SD5-XCU-ERN-79 is applied, restoring the horizontal lines to XCU Table 4-1, Expressions in Decreasing Precedence in *awk*, and SD5-XCU-ERN-80 is applied, changing the order of some table entries.

basename

Purpose: Return non-directory portion of a pathname.

Synopsis: `basename string [suffix]`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

batch

Purpose: Schedule commands to be executed in a batch queue.

Synopsis: `batch`

Derivation: First released in Issue 2.

Issue 7: The *batch* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced by the *batch* utility.

bc

Purpose: Arbitrary-precision arithmetic language.

Synopsis: `bc [-l] [file...]`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

bg

Purpose: Run jobs in the background.

UP

Synopsis: `bg [job_id...]`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

c99

Purpose: Compile standard C programs.

CD

Synopsis: `c99 [options...] pathname [[pathname] [-I directory] [-L directory] [-l library]]...`

Derivation: First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #020 (SD5-XCU-ERN-10) is applied, adding a statement to the OUTPUT FILES section about unspecified behavior when the pathname of an object file or executable file to be created by *c99* resolves to an existing directory entry for a file that is not a regular file.

Austin Group Interpretation 1003.1-2001 #166 is applied, adding information about the use of *getconf* to obtain *c99* arguments used for the threaded programming environment.

Austin Group Interpretation 1003.1-2001 #190 is applied, clarifying the handling of trailing white-space characters.

Austin Group Interpretation 1003.1-2001 #191 is applied, adding APPLICATION USAGE and RATIONALE regarding C-language trigraphs.

SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply (options can be interspersed with operands).

SD5-XCU-ERN-11 is applied, adding the `<net/if.h>` header to the descriptions of `-l c` and `-l xnet`.

SD5-XCU-ERN-65 is applied, updating the EXAMPLES section.

The *getconf* variables for the supported programming environments are updated to be V7.

The `-l trace` library is marked obsolescent.

The *c99* reference page is rewritten to describe `-l` as an option rather than an operand.

cal

Purpose: Print a calendar.

XSI Synopsis: `cal [[month] year]`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

cat

Purpose: Concatenate and print files.

Synopsis: `cat [-u] [file...]`

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-174 is applied, changing the RATIONALE concerning an alternative to the historical *cat -etv*.

cd

Purpose: Change the working directory.

Synopsis: `cd [-L|-P] [directory]`
`cd -`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #037 is applied, updating steps 6 through 10 of the processing performed by *cd* to correct a number of defects.

Austin Group Interpretation 1003.1-2001 #199 is applied, clarifying how the *cd* utility handles concatenation of two pathnames when the first pathname ends in a slash character.

Step 7 of the processing performed by *cd* is revised to refer to **curpath** instead of “the operand”.

The description of how the *cd* utility sets the *PWD* environment variable has been changed to refer to the output of the *pwd* utility.

cflow

Purpose: Generate a C-language flowgraph (**DEVELOPMENT**).

XSI Synopsis: `cflow [-r] [-d num] [-D name[=def]]... [-i incl] [-I dir]... [-U dir]... file...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

chgrp

Purpose: Change the file group ownership.

Synopsis: `chgrp [-h] group file...`
`chgrp -R [-H|-L|-P] group file...`

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-8 is applied, removing the **-R** from the first line of the SYNOPSIS.

chmod

Purpose: Change the file modes.

Synopsis: `chmod [-R] mode file...`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #130 is applied, adding text to the DESCRIPTION about marking for update the last file status change timestamp of the file.

chown

Purpose: Change the file ownership.

Synopsis: `chown [-h] owner[:group] file...`
`chown -R [-H|-L|-P] owner[:group] file...`

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-9 is applied, removing the **-R** from the first line of the SYNOPSIS. The description of the **-h** and **-P** options is revised.

cksum

Purpose: Write file checksums and sizes.

Synopsis: `cksum [file...]`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is '-' and the implementation treats the '-' as meaning standard input.

cmp

Purpose: Compare two files.

Synopsis: `cmp [-l|-s] file1 file2`

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-96 is applied, updating the STDERR section to specify the output when the **-l** option is used.

comm

Purpose: Select or reject lines common to two files.

Synopsis: `comm [-123] file1 file2`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

command

Purpose: Execute a simple command.

Synopsis: `command [-p] command_name [argument...]`

`command [-p][-v|-V] command_name`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #196 is applied, changing the SYNOPSIS to allow `-p` to be used with `-v` (or `-V`).

The `command` utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

The APPLICATION USAGE and EXAMPLES are revised to replace the non-standard `getconf_CS_PATH` with `getconf PATH`.

compress

Purpose: Compress data.

XSI Synopsis: `compress [-fv] [-b bits] [file...]`
`compress [-cfv] [-b bits] [file]`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #125 is applied, revising the ENVIRONMENT VARIABLES section in relation to locale usage.

cp

Purpose: Copy files.

Synopsis: `cp [-Pfp] source_file target_file`

`cp [-Pfp] source_file... target`

`cp -R [-H|-L|-P] [-fip] source_file... target`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, specifying that a `source_file` or `target_file` operand of `'-'` shall refer to a file named `'-'`; implementations shall not treat them as meaning standard input or standard output.

Austin Group Interpretation 1003.1-2001 #164 is applied, making the behavior unspecified when `cp` encounters an existing `dest_file` that was written by a previous step.

Austin Group Interpretation 1003.1-2001 #165 is applied, correcting the description

of the `-i` option to reflect that prompts are not written for existing directory files (only non-directory files), as per the detailed steps in the DESCRIPTION.

Austin Group Interpretation 1003.1-2001 #168 is applied, updating the description of how two pathnames are concatenated so that a slash character is only inserted if the first pathname does not end in a slash.

The obsolescent `-r` option is removed.

The `-P` option is added to the SYNOPSIS and to the DESCRIPTION with respect to its use without the `-R` option.

crontab

Purpose: Schedule periodic background work.

Synopsis: `crontab [file]`

UP

```
crontab [-e|-l|-r]
```

Derivation: First released in Issue 2.

Issue 7: The `crontab` utility (except for the `-e` option) is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced by the `crontab` utility.

The first example is changed to remove the unreliable use of `find | xargs`.

csplit

Purpose: Split files based on context.

Synopsis: `csplit [-ks] [-f prefix] [-n number] file arg...`

Derivation: First released in Issue 2.

Issue 7: The `csplit` utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

The SYNOPSIS and OPERANDS sections are revised to clarify that use of a single `arg` operand is permitted.

ctags

Purpose: Create a tags file (**DEVELOPMENT, FORTRAN**).

SD

Synopsis: `ctags [-a] [-f tagsfile] pathname...`

```
ctags -x pathname...
```

Derivation: First released in Issue 4.

Issue 7: The `ctags` utility is no longer dependent on support for the User Portability Utilities option.

cut

Purpose: Cut out selected fields of each line of a file.

Synopsis: `cut -b list [-n] [file...]`
`cut -c list [file...]`
`cut -f list [-d delim] [-s] [file...]`

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-171 is applied, adding APPLICATION USAGE regarding the use of the *cut* and *fold* utilities to create text files out of files with arbitrary line lengths.

cxref

Purpose: Generate a C-language program cross-reference table (**DEVELOPMENT**).

XSI Synopsis: `cxref [-cs] [-o file] [-w num] [-D name[=def]]... [-I dir]... [-U name]... file...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

date

Purpose: Write the date and time.

XSI Synopsis: `date [-u] [+format]`
`date [-u] mmdhmm[[cc]yy]`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

dd

Purpose: Convert and copy a file.

Synopsis: `dd [operand...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #102 is applied, adding requirements for the output file to be extended when the input file is empty, **seek=expr** is specified but **conv=notrunc** is not, and either the size of the seek is greater than the previous size of the output file or the output file did not previously exist.

delta

Purpose: Make a delta (change) to an SCCS file (**DEVELOPMENT**).

XSI Synopsis: `delta [-nps] [-g list] [-m mrlist] [-r SID] [-y[comment]] file...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

df

Purpose: Report free disk space.

XSI Synopsis: `df [-k] [-P|-t] [file...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #099 is applied, clarifying the XSI requirements for operands which name a special file containing a file system.

The *df* utility is removed from the User Portability Utilities option. User Portability Utilities is now an option for interactive utilities.

diff

Purpose: Compare two files.

Synopsis: `diff [-c|-e|-f|-u|-C n|-U n] [-br] file1 file2`

When the `-u` option is specified, *diff* produces output in a form that provides three lines of unified context.

When the `-U n` option is specified, *diff* produces output in a form that provides *n* lines of unified context.

The `-u` or `-U` options behave like the `-c` or `-C` options, except that the context lines are not repeated; instead, the context, deleted, and added lines are shown together, interleaved.

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #114 is applied, requiring *diff* to detect infinite loops in the file system when the `-r` option is specified.

Austin Group Interpretation 1003.1-2001 #115 is applied, updating requirements when block or character special files are encountered in directories being compared.

Austin Group Interpretation 1003.1-2001 #192 is applied, clarifying the behavior if one or both files are non-text files.

SD5-XCU-ERN-103 and SD5-XCU-ERN-120 are applied, adding the `-u` and `-U` options.

dirname

Purpose: Return the directory portion of a pathname.

Synopsis: `dirname string`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

du

Purpose: Estimate file space usage.

Synopsis: `du [-a|-s] [-kx] [-H|-L] [file...]`

Derivation: First released in Issue 2.

Issue 7: The *du* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

echo

Purpose: Write arguments to standard output.

Synopsis: `echo [string...]`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

ed

Purpose: Edit text.

Synopsis: `ed [-p string] [-s] [file]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if an operand is '-'.
Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BRE back-references when a subexpression does not participate in the match.

SD5-XCU-ERN-94 is applied, updating text in the EXTENDED DESCRIPTION where a terminal disconnect is detected (in Commands in *ed*).

env

Purpose: Set the environment for command invocation.

Synopsis: `env [-i] [name=value]... [utility [argument...]]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first argument is '-'.
Austin Group Interpretation 1003.1-2001 #047 is applied, providing RATIONALE on how to use the *env* utility to preserve a conforming environment.

The EXAMPLES section is revised to change the use of *env -i* so that it preserves a conforming environment.

ex

Purpose: Text editor.

UP

Synopsis: `ex [-rR] [-s|-v] [-c command] [-t tagstring]
[-w size] [file...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if an operand is '-'.
Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BRE back-references when a subexpression does not participate in the match.

expand

Purpose: Convert tabs to spaces.

Synopsis: `expand [-t tablist] [file...]`

Derivation: First released in Issue 4.

Issue 7: The *expand* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

expr

Purpose: Evaluate arguments as an expression.

Synopsis: `expr operand...`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BRE back-references when a subexpression does not participate in the match.

false

Purpose: Return false value.

Synopsis: `false`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

fc

Purpose: Process the command history list.

UP

Synopsis: `fc [-r] [-e editor] [first [last]]
fc -l [-nr] [first [last]]
fc -s [old=new] [first]`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

fg

Purpose: Run jobs in the foreground.

UP Synopsis: `fg [job_id]`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

file

Purpose: Determine file type.

Synopsis: `file [-dh] [-M file] [-m file] file...`

`file -i [-h] file...`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is used if a *file* operand is '-' and the implementation treats the '-' as meaning standard input.

SD5-XCU-ERN-4 is applied, adding further entries in the Notes column in XCU Table 4-9, File Utility Output Strings .

The *file* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

The EXAMPLES section is revised to make use of the "--" delimiter.

find

Purpose: Find files.

Synopsis: `find [-H|-L] path... [operand_expression...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #127 is applied, rephrasing the description of the **-exec** primary to be "immediately follows".

Austin Group Interpretation 1003.1-2001 #185 is applied, clarifying the requirements for the **-H** and **-L** options.

Austin Group Interpretation 1003.1-2001 #186 is applied, clarifying the requirements for the evaluation of *path* operands with trailing slashes.

Austin Group Interpretation 1003.1-2001 #195 is applied, clarifying the interpretation of the first operand.

SD5-XCU-ERN-48 is applied, clarifying the **-L** option in the case that the file referenced by a symbolic link does not exist.

SD5-XCU-ERN-117 is applied, clarifying the **-perm** primary.

SD5-XCU-ERN-122 is applied, adding a new EXAMPLE showing the useful technique:

```
-exec sh -c '... "$@" ...' sh { } +
```

The description of the **-name** primary is revised and the **-path** primary is added (with a new example).

fold

Purpose: Filter for folding lines.

Synopsis: `fold [-bs] [-w width] [file...]`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is '-' and the implementation treats the '-' as meaning standard input.

Austin Group Interpretation 1003.1-2001 #204 is applied, updating the DESCRIPTION to clarify when a <newline> can be inserted before or after a <backspace>.

fort77

Purpose: FORTRAN compiler (**FORTTRAN**).

FD Synopsis: `fort77 [-c] [-g] [-L directory]... [-O optlevel] [-o outfile] [-s] [-w] operand...`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

fuser

Purpose: List process IDs of all processes that have one or more files open.

XSI Synopsis: `fuser [-cfu] file...`

Derivation: First released in Issue 5.

Issue 7: SD5-XCU-ERN-90 is applied, updating the EXAMPLES section.

gencat

Purpose: Generate a formatted message catalog.

Synopsis: `gencat catfile msgfile...`

Derivation: First released in Issue 3.

Issue 7: The *gencat* utility is moved from the XSI option to the Base.

get

Purpose: Get a version of an SCCS file (**DEVELOPMENT**).

XSI Synopsis: `get [-begkmnlLpst] [-c cutoff] [-i list] [-r SID] [-x list] file...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

getconf

Purpose: Get configuration values.

Synopsis: `getconf [-v specification] system_var`
`getconf [-v specification] path_var pathname`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

getopts

Purpose: Parse utility options.

Synopsis: `getopts optstring name [arg...]`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

grep

Purpose: Search a file for a pattern.

Synopsis: `grep [-E|-F] [-c|-l|-q] [-insvx] -e pattern_list`
`[-e pattern_list]... [-f pattern_file]... [file...]`
`grep [-E|-F] [-c|-l|-q] [-insvx] [-e pattern_list]...`
`-f pattern_file [-f pattern_file]... [file...]`
`grep [-E|-F] [-c|-l|-q] [-insvx] pattern_list [file...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is '-' and the implementation treats the '-' as meaning standard input.

SD5-XCU-ERN-98 is applied, updating the STDOUT section to reflect the fact that the -I and -q options are shown in the SYNOPSIS as mutually exclusive.

hash

Purpose: Remember or report utility locations.

Synopsis: `hash [utility...]`
`hash -r`

Derivation: First released in Issue 2.

Issue 7: The *hash* utility is moved from the XSI option to the Base.

head

Purpose: Copy the first part of files.

Synopsis: `head [-n number] [file...]`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is '-' and the implementation treats the '-' as meaning standard input.

The EXAMPLES section is revised to make use of the "--" delimiter.

iconv

Purpose: Codeset conversion.

Synopsis: `iconv [-cs] -f frommap -t tomap [file...]`
`iconv -f fromcode [-cs] [-t tocode] [file...]`
`iconv -t tocode [-cs] [-f fromcode] [file...]`
`iconv -l`

Derivation: First released in Issue 3.

Issue 7: No functional changes are made in this issue.

id

Purpose: Return user identity.

Synopsis: `id [user]`
`id -G [-n] [user]`
`id -g [-nr] [user]`
`id -u [-nr] [user]`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

ipcrm

Purpose: Remove an XSI message queue, semaphore set, or shared memory segment identifier.

XSI Synopsis: `ipcrm [-q msgid|-Q msgkey|-s semid|-S semkey|-m shmid|-M shmkey]...`

Derivation: First released in Issue 5.

Issue 7: No functional changes are made in this issue.

ipcs

Purpose: Report XSI interprocess communication facilities status.

XSI Synopsis: `ipcs [-qms] [-a|-bcopt]`

Derivation: First released in Issue 5.

Issue 7: No functional changes are made in this issue.

jobs

Purpose: Display status of jobs in the current session.

UP Synopsis: `jobs [-l|-p] [job_id...]`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

join

Purpose: Relational database operator.

Synopsis: `join [-a file_number|-v file_number] [-e string] [-o list]
[-t char] [-1 field] [-2 field] file1 file2`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

kill

Purpose: Terminate or signal processes.

Synopsis: `kill -s signal_name pid...`

`kill -l [exit_status]`

XSI `kill [-signal_name] pid...`

`kill [-signal_number] pid...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

lex

Purpose: Generate programs for lexical tasks (**DEVELOPMENT**).

CD Synopsis: `lex [-t] [-n|-v] [file...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #190 is applied, clarifying the requirements for generated code to conform to the ISO C standard.

Austin Group Interpretation 1003.1-2001 #191 is applied, clarifying the handling of C-language trigraphs and curly brace preprocessing tokens.

SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply (options can be interspersed with operands).

link

Purpose: Call *link()* function.

XSI Synopsis: `link file1 file2`

Derivation: First released in Issue 5.

Issue 7: No functional changes are made in this issue.

ln

Purpose: Link files.

Synopsis: `ln [-fs] [-L|-P] source_file target_file`
`ln [-fs] [-L|-P] source_file... target_dir`

When the **-L** option is specified (and the **-s** option is not specified), for each *source_file* operand that names a file of type symbolic link, *ln* creates a (hard) link to the file referenced by the symbolic link.

When the **-P** option is specified (and the **-s** option is not specified), for each *source_file* operand that names a file of type symbolic link, *ln* creates a (hard) link to the symbolic link itself.

If the **-s** option is not specified and neither a **-L** nor a **-P** option is specified, it is implementation-defined which of the **-L** and **-P** options will be used as the default.

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #164 is applied, allowing *ln* to report an error when it encounters an existing destination path that was written by a previous step.

Austin Group Interpretation 1003.1-2001 #168 is applied, updating the description of how two pathnames are concatenated so that a slash character is only inserted if the first pathname does not end in a slash.

Austin Group Interpretation 1003.1-2001 #169 is applied, updating the requirements when *destination* names the same directory entry as the current *source_file*.

The **-L** and **-P** options are added to provide control over how the *ln* utility creates hard links to symbolic links.

locale

Purpose: Get locale-specific information.

Synopsis: `locale [-a|-m]`
`locale [-ck] name...`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #017 is applied, clarifying the standard output for the **-k** option for non-numeric compound keyword values.

Austin Group Interpretations 1003.1-2001 #021 and #088 are applied, clarifying the standard output for the `-k` option when `LANG` is not set or is an empty string.

localedef

Purpose: Define locale environment.

Synopsis: `localedef [-c] [-f charmap] [-i sourcefile]
[-u code_set_name] name`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

logger

Purpose: Log messages.

Synopsis: `logger string...`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

logname

Purpose: Return the user's login name.

Synopsis: `logname`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

lp

Purpose: Send files to a printer.

Synopsis: `lp [-c] [-d dest] [-n copies] [-msw] [-o option]...
[-t title] [file...]`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

ls

Purpose: List directory contents.

XS1 Synopsis: `ls [-ACFRSacdfiklmnpqrstuxl] [-H|-L] [-go] [file...]`

When the `-A` option is specified, `ls` writes out all directory entries, including those whose names begin with a <period> (`'.'`) but excluding the entries `dot` and `dot-dot` (if they exist).

When the `-S` option is specified, `ls` sorts with the primary key being file size (in decreasing order) and the secondary key being filename in the collating sequence (in increasing order).

When the `-k` option is specified, `ls` sets the block size for the `-s` option and the per-directory block count written for the `-l`, `-n`, `-s`, `-g`, and `-o` options to 1 024 bytes.

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #101 is applied, clarifying the optional alternate access method flag in the STDOUT section.

Austin Group Interpretation 1003.1-2001 #128 is applied, clarifying the DESCRIPTION and the definition of the **-R** option with regard to symbolic links.

Austin Group Interpretation 1003.1-2001 #198 is applied, clarifying the requirements for the **-H** option for symbolic links specified on the command line.

SD5-XCU-ERN-50 is applied, adding the **-A** option.

The **-S** option is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The **-f**, **-m**, **-n**, **-p**, **-s**, and **-x** options are moved from the XSI option to the Base.

The description of the **-f**, **-s**, and **-t** options are revised and the **-k** option is added.

m4

Purpose: Macro processor.

Synopsis: `m4 [-s] [-D name[=val]]... [-U name]... file...`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #117 is applied, marking the **maketemp** macro obsolescent and adding a new **mkstemp** macro.

Austin Group Interpretation 1003.1-2001 #207 is applied, clarifying the handling of white-space characters that precede or trail any macro arguments.

SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply (options can be interspersed with operands).

SD5-XCU-ERN-99 is applied, clarifying the definition of the **divert** macro in the EXTENDED DESCRIPTION.

SD5-XCU-ERN-100 is applied, clarifying the definition of the **syscmd** macro in the EXTENDED DESCRIPTION.

SD5-XCU-ERN-101 is applied, clarifying the definition of the **undivert** macro in the EXTENDED DESCRIPTION.

SD5-XCU-ERN-111 is applied to the EXTENDED DESCRIPTION, clarifying that the string "**`\${**" produces unspecified behavior.

SD5-XCU-ERN-112 is applied, updating the **changequote** macro.

SD5-XCU-ERN-118 is applied, clarifying the definition of the **define** macro in the EXTENDED DESCRIPTION and APPLICATION USAGE sections.

SD5-XCU-ERN-119 is applied, clarifying the definition of the **translit** macro in the EXTENDED DESCRIPTION and RATIONALE sections.

SD5-XCU-ERN-130 is applied, making the behavior unspecified when macro names are used without arguments.

SD5-XCU-ERN-131 is applied, making the behavior unspecified when either argument to the **changecom** macro is provided but null.

SD5-XCU-ERN-137 is applied, updating the description of the **eval** macro in the EXTENDED DESCRIPTION and APPLICATION USAGE sections.

The *m4* utility is moved from the XSI option to the Base.

mailx

Purpose: Process messages.

Synopsis: **Send Mode**

```
mailx [-s subject] address...
```

Receive Mode

UP

```
mailx -e
mailx [-HiNn] [-F] [-u user]
mailx -f [-HiNn] [-F] [file]
```

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #089 is applied, clarifying the effect of the *LC_TIME* environment variable.

Austin Group Interpretation 1003.1-2001 #090 is applied, updating the description of the **next** command.

make

Purpose: Maintain, update, and regenerate groups of programs (**DEVELOPMENT**).

SD

Synopsis: `make [-einpqrst] [-f makefile]... [-k|-S] [macro=value...]
[target_name...]`

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply (options can be interspersed with operands).

Include lines in makefiles are introduced.

Austin Group Interpretation 1003.1-2001 #131 is applied, changing the **Makefile Execution** section.

man

Purpose: Display system documentation.

Synopsis: `man [-k] name...`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #108 is applied, clarifying that informational messages may appear on standard error.

mesg

Purpose: Permit or deny messages.

Synopsis: `mesg [y|n]`

Derivation: First released in Issue 2.

Issue 7: The *mesg* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

mkdir

Purpose: Make directories.

Synopsis: `mkdir [-p] [-m mode] dir...`

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-56 is applied, aligning the `-m` option with the IEEE P1003.2b draft standard to clarify an ambiguity.

mkfifo

Purpose: Make FIFO special files.

Synopsis: `mkfifo [-m mode] file...`

Derivation: First released in Issue 3.

Issue 7: No functional changes are made in this issue.

more

Purpose: Display files on a page-by-page basis.

UP Synopsis: `more [-ceisu] [-n number] [-p command] [-t tagstring]
[file...]`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized as an option delimiter in the OPTIONS section.

mv

Purpose: Move files.

Synopsis: `mv [-if] source_file target_file`
`mv [-if] source_file... target_dir`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #016 is applied, updating requirements relating to a *target_file* operand with a trailing slash .

Austin Group Interpretation 1003.1-2001 #164 is applied, allowing *mv* to report an error when it encounters an existing destination path that was written by a previous step.

Austin Group Interpretation 1003.1-2001 #168 is applied, updating the description of how two pathnames are concatenated so that a slash character is only inserted

if the first pathname does not end in a slash.

Austin Group Interpretation 1003.1-2001 #169 is applied, updating the requirements when the *source_file* operand and destination path name the same existing file.

SD5-XCU-ERN-51 is applied to the DESCRIPTION, clarifying that it is unspecified whether hard links to other files are preserved when files are being duplicated to another file system.

Changes are made related to support for finegrained timestamps.

newgrp

Purpose: Change to a new group.

Synopsis: `newgrp [-l] [group]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first argument is `'-'`.

The *newgrp* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

nice

Purpose: Invoke a utility with an altered nice value.

Synopsis: `nice [-n increment] utility [argument...]`

Derivation: First released in Issue 4.

Issue 7: The *nice* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

nl

Purpose: Line numbering filter.

XSI Synopsis: `nl [-p] [-b type] [-d delim] [-f type] [-h type] [-i incr] [-l num] [-n format] [-s sep] [-v startnum] [-w width] [file]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is `'-'` and the implementation treats the `'-'` as meaning standard input.

nm

Purpose: Write the name list of an object file (**DEVELOPMENT**).

SD XSI Synopsis: `nm [-APv] [-g|-u] [-t format] file...`
`nm [-APv] [-efox] [-g|-u] [-t format] file...`

Derivation: First released in Issue 2.

Issue 7: The *nm* utility is removed from the User Portability Utilities option. User Portability Utilities is now an option for interactive utilities.

nohup

Purpose: Invoke a utility immune to hangups.

Synopsis: `nohup utility [argument...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #104 is applied, allowing *nohup* to redirect standard input from an unspecified file if it is associated with a terminal.

Austin Group Interpretations 1003.1-2001 #105 and #106 are applied, updating requirements related to redirection of standard output and standard error.

od

Purpose: Dump files in various formats.

Synopsis: `od [-v] [-A address_base] [-j skip] [-N count]
[-t type_string]... [file...]`

XSI

```
od [-bcdosx] [file] [[+]offset[.][b]]
```

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is '-' and the implementation treats the '-' as meaning standard input.

paste

Purpose: Merge corresponding or subsequent lines of files.

Synopsis: `paste [-s] [-d list] file...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

patch

Purpose: Apply changes to files.

Synopsis: `patch [-blNR] [-c|-e|-n|-u] [-d dir] [-D define] [-i patchfile]
[-o outfile] [-p num] [-r rejectfile] [file]`

When the *-u* option is specified, *patch* interprets the patch file as a unified context difference (the output of the *diff* utility when the *-u* or *-U* options are specified).

Derivation: First released in Issue 4.

Issue 7: The *patch* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-103 and SD5-XCU-ERN-120 are applied, adding the *-u* option.

Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the *LC_CTYPE* environment variable and adding the *LC_COLLATE* environment

variable.

pathchk

Purpose: Check pathnames.

Synopsis: `pathchk [-p] [-P] pathname...`

When the **-P** option is specified, *pathchk* writes a diagnostic for each *pathname* operand that:

- Contains a component whose first character is the <hyphen> character
- Is empty

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretations 1003.1-2001 #039 and #040 are applied, adding the **-P** option.

SD5-XCU-ERN-121 is applied, updating the way *xargs* is used in the EXAMPLES section.

pax

Purpose: Portable archive interchange.

Synopsis: `pax [-dv] [-c|-n] [-H|-L] [-o options] [-f archive]
[-s replstr]... [pattern...]`

`pax -r[-c|-n] [-dikuv] [-H|-L] [-f archive] [-o options]...
[-p string]... [-s replstr]... [pattern...]`

`pax -w [-dituvX] [-H|-L] [-b blocksize] [[-a] [-f archive]]
[-o options]... [-s replstr]... [-x format] [file...]`

`pax -r -w [-diklntuvX] [-H|-L] [-o options]... [-p string]...
[-s replstr]... [file...] directory`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #011 is applied, clarifying how symbolic links are archived in **cpio** format.

Austin Group Interpretation 1003.1-2001 #086 is applied, clarifying that when a list of files to copy is read from the standard input, each entry in the list is processed as if it had been a *file* operand on the command line.

Austin Group Interpretation 1003.1-2001 #109 is applied, adding the **hdrcharset** keyword to the *pax* extended headers, and related requirements.

SD5-XCU-ERN-2 is applied, making the **-c** and **-n** options mutually-exclusive in the SYNOPSIS.

SD5-XCU-ERN-60 is applied, revising text which incorrectly implied that the **-x** option could be used in copy mode.

The *pax* utility is no longer allowed to create separate identical symbolic links when extracting linked symbolic links from an archive, because the standard now requires implementations to support (hard) linking of symbolic links.

pr

Purpose: Print files.

Synopsis: `pr [+page] [-column] [-adFmrt] [-e[char][gap]] [-h header]
 [-i[char][gap]] [-l lines] [-n[char][width]]
 XSI [-o offset] [-s[char]] [-w width] [-fp] [file...]`

Derivation: First released in Issue 2.

Issue 7: PASC Interpretation 1003.2-92 #151 (SD5-XCU-ERN-44) is applied, replacing “two or more” in the description of the `-i` option with “one or more”.

Austin Group Interpretation 1003.1-2001 #093 is applied, adding APPLICATION USAGE warning that a first operand that starts with a <plus-sign> needs to be preceded with the “--” argument that denotes the end of the options.

printf

Purpose: Write formatted output.

Synopsis: `printf format [argument...]`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #175 is applied, updating requirements related to floating-point conversions to align with the `printf()` function.

Austin Group Interpretation 1003.1-2001 #177 is applied, clarifying the behavior of the `%c` conversion.

prs

Purpose: Print an SCCS file (**DEVELOPMENT**).

XSI Synopsis: `prs [-a] [-d dataspec] [-r[SID]] file...
 prs [-e|-l] -c cutoff [-d dataspec] file...
 prs [-e|-l] -r[SID] [-d dataspec] file...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

ps

Purpose: Report process status.

XSI Synopsis: `ps [-aA] [-defl] [-g grouplist] [-G grouplist]
 [-n namelist] [-o format]... [-p proclist] [-t termlist]
 [-u userlist] [-U userlist]`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

pwd

Purpose: Return working directory name.

Synopsis: `pwd [-L|-P]`

Derivation: First released in Issue 2.

Issue 7: Changes have been made to match the changes to the `getcwd()` function, adding text to address the case where the current directory is deeper in the file hierarchy than `{PATH_MAX}` bytes, and adding the requirements relating to pathnames beginning with two slash characters.

qalter

Purpose: Alter batch job.

OB BE Synopsis: `qalter [-a date_time] [-A account_string] [-c interval] [-e path_name] [-h hold_list] [-j join_list] [-k keep_list] [-l resource_list] [-m mail_options] [-M mail_list] [-N name] [-o path_name] [-p priority] [-r y|n] [-S path_name_list] [-u user_list] job_identifier...`

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The `qalter` utility is marked obsolescent.

qdel

Purpose: Delete batch jobs.

OB BE Synopsis: `qdel job_identifier...`

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The `qdel` utility is marked obsolescent.

qhold

Purpose: Hold batch jobs.

OB BE Synopsis: `qhold [-h hold_list] job_identifier...`

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The `qhold` utility is marked obsolescent.

qmove

Purpose: Move batch jobs.

OB BE Synopsis: `qmove destination job_identifier...`

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The `qmove` utility is marked obsolescent.

qmsg

Purpose: Send message to batch jobs.

OB BE Synopsis: `qmsg [-EO] message_string job_identifer...`

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The *qmsg* utility is marked obsolescent.

qrerun

Purpose: Rerun batch jobs.

OB BE Synopsis: `qrerun job_identifer...`

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The *qrerun* utility is marked obsolescent.

qrls

Purpose: Release batch jobs.

OB BE Synopsis: `qrls [-h hold_list] job_identifer...`

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The *qrls* utility is marked obsolescent.

qselect

Purpose: Select batch jobs.

OB BE Synopsis: `qselect [-a [op]date_time] [-A account_string]
[-c [op]interval] [-h hold_list] [-l resource_list]
[-N name] [-p [op]priority] [-q destination]
[-r y|n] [-s states] [-u user_list]`

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The *qselect* utility is marked obsolescent.

qsig

Purpose: Signal batch jobs.

OB BE Synopsis: `qsig [-s signal] job_identifer...`

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The *qsig* utility is marked obsolescent.

qstat

Purpose: Show status of batch jobs.

OB BE Synopsis: `qstat [-f] job_identifier...`
`qstat -Q [-f] destination...`
`qstat -B [-f] server_name...`

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The *qstat* utility is marked obsolescent.

qsub

Purpose: Submit a script.

OB BE Synopsis: `qsub [-a date_time] [-A account_string] [-c interval]`
`[-C directive_prefix] [-e path_name] [-h] [-j join_list]`
`[-k keep_list] [-m mail_options] [-M mail_list] [-N name]`
`[-o path_name] [-p priority] [-q destination] [-r y|n]`
`[-S path_name_list] [-u user_list] [-v variable_list] [-V]`
`[-z] [script]`

Derivation: Derived from IEEE Std 1003.2d-1994.

Issue 7: The *qsub* utility is marked obsolescent.

read

Purpose: Read a line from standard input.

Synopsis: `read [-r] var...`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #194 is applied, clarifying the handling of the <backslash> escape character.

renice

Purpose: Set nice values of running processes.

Synopsis: `renice [-g|-p|-u] -n increment ID...`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply (options can be interspersed with operands).

The *renice* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

rm

Purpose: Remove directory entries.

Synopsis: `rm [-fiRr] file...`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #019 is applied, requiring *rm* to report an error if an operand resolves to the root directory.

Austin Group Interpretation 1003.1-2001 #091 is applied, updating the description of exit status 0 in the EXIT STATUS section.

rmdel

Purpose: Remove a delta from an SCCS file (**DEVELOPMENT**).

XS1 Synopsis: `rmdel -r SID file...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

rmdir

Purpose: Remove directories.

Synopsis: `rmdir [-p] dir...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

sact

Purpose: Print current SCCS file-editing activity (**DEVELOPMENT**).

XS1 Synopsis: `sact file...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

sccs

Purpose: Front end for the SCCS subsystem (**DEVELOPMENT**).

XS1 Synopsis: `sccs [-r] [-d path] [-p path] command [options...] [operands...]`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

sed

Purpose: Stream editor.

Synopsis: `sed [-n] script [file...]`
`sed [-n] -e script [-e script]... [-f script_file]... [file...]`
`sed [-n] [-e script]... -f script_file [-f script_file]... [file...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BRE back-references when a subexpression does not participate in the match.

Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is '-' and the implementation treats the '-' as meaning standard input.

SD5-XCU-ERN-123 is applied, updating the SYNOPSIS so that it correctly reflects the relationship between the *script* operand and the -e and -f options.

A second example is added, giving a simpler method of squeezing empty lines.

sh

Purpose: Shell, the standard command language interpreter.

Synopsis: `sh [-abCefhimnuvx] [-o option]... [+abCefhimnuvx] [+o option]... [command_file [argument...]]`
`sh -c [-abCefhimnuvx] [-o option]... [+abCefhimnuvx] [+o option]... command_string [command_name [argument...]]`
`sh -s [-abCefhimnuvx] [-o option]... [+abCefhimnuvx] [+o option]... [argument...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #098 is applied, changing the description of *IFS* to match the one in section 2.5.3.

The description of the *PWD* environment variable is updated to reflect that assignments to the variable may always be ignored.

Minor changes are made to the install script example in the APPLICATION USAGE section.

sleep

Purpose: Suspend execution for an interval.

Synopsis: `sleep time`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

sort

Purpose: Sort, merge, or sequence check text files.

Synopsis: `sort [-m] [-o output] [-bdfinru] [-t char] [-k keydef]...
[file...]`

`sort [-c|-C] [-bdfinru] [-t char] [-k keydef] [file]`

The `-C` option is the same as `-c`, except that a warning message is not sent to standard error if disorder or, with `-u`, a duplicate key is detected.

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply (options can be interspersed with operands), and noting that '+' may be recognized as an option delimiter.

Austin Group Interpretation 1003.1-2001 #120 is applied, updating the `-c` option to require that the warning message sent to standard error indicates where the disorder or duplicate key was found, and introducing the `-C` option.

XCU-ERN-81 is applied, modifying the description of the `-i` option to state that the behavior is undefined for a sort key for which `-n` also applies.

split

Purpose: Split files into pieces.

Synopsis: `split [-l line_count] [-a suffix_length] [file[name]]
split -b n[k|m] [-a suffix_length] [file[name]]`

Derivation: First released in Issue 2.

Issue 7: The `split` utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

strings

Purpose: Find printable strings in files.

Synopsis: `strings [-a] [-t format] [-n number] [file...]`

Derivation: First released in Issue 4.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first argument is '-'. .

The `strings` utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

strip

Purpose: Remove unnecessary information from strippable files (**DEVELOPMENT**).

sd Synopsis: `strip file...`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #103 is applied, clarifying that XSI-conformant systems support use of `strip` on archive files containing object files or relocatable files.

stty

Purpose: Set the options for a terminal.

Synopsis: `stty [-a|-g]`
`stty operand...`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #144 is applied, moving functionality relating to the IXANY symbol from the XSI option to the Base.

tabs

Purpose: Set terminal tabs.

XSI Synopsis: `tabs [-n|-a|-a2|-c|-c2|-c3|-f|-p|-s|-u] [-T type]`
`tabs [-T type] n[[sep[+]n]...]`

Derivation: First released in Issue 2.

Issue 7: The `tabs` utility is removed from the User Portability Utilities option. User Portability Utilities is now an option for interactive utilities.

tail

Purpose: Copy the last part of a file.

Synopsis: `tail [-f] [-c number|-n number] [file]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized as an option delimiter in the OPTIONS section.

Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if the `file` operand is '-' and the implementation treats the '-' as meaning standard input.

Austin Group Interpretation 1003.1-2001 #100 is applied, adding the requirement on applications that if the sign of the option-argument `number` is '+', the `number` option-argument is non-zero.

SD5-XCU-ERN-114 is applied, updating the `-f` option so that a FIFO on standard input is treated the same as a pipe.

talk

Purpose: Talk to another user.

UP Synopsis: `talk address [terminal]`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

tee

Purpose: Duplicate standard input.

Synopsis: `tee [-ai] [file...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, specifying that a *file* operand of '-' shall refer to a file named '-'; implementations shall not treat it as meaning standard output.

test

Purpose: Evaluate expression.

Synopsis: `test [expression]`
`[[expression]]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #107 is applied, marking the XSI extensions specifying the `-a` and `-o` primaries and the `'(' and ')'` operators as obsolescent. Applications should combine separate *test* commands instead. For example, using:

```
test expr1 && test expr2
```

instead of:

```
test expr1 -a expr2
```

time

Purpose: Time a simple command.

Synopsis: `time [-p] utility [argument...]`

Derivation: First released in Issue 2.

Issue 7: The *time* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

touch

Purpose: Change file access and modification times.

Synopsis: `touch [-acm] [-r ref_file|-t time|-d date_time] file...`

When the `-d date_time` option is specified, *touch* uses the specified *date_time* instead of the current time. The option-argument is a string of the form:

```
YYYY-MM-DDThh:mm:SS[.frac][tz]
```

or:

```
YYYY-MM-DDThh:mm:SS[,frac][tz]
```

where:

- YYYY are at least four decimal digits giving the year.

- *MM*, *DD*, *hh*, *mm*, and *SS* are as with `-t time`.
- *T* is the time designator, and can be replaced by a single `<space>`.
- [*.frac*] and [*,frac*] are either empty, or a `<period>` (`'.'`) or a `<comma>` (`','`) respectively, followed by one or more decimal digits, specifying a fractional second.
- [*tz*] is either empty, signifying local time, or the letter `'Z'`, signifying UTC.

The following examples demonstrate the use of the `-d` option.

Create or update a file called **dwc**; the resulting file has both the last data modification and last data access timestamps set to November 12, 2007 at 10:15:30 local time:

```
touch -d 2007-11-12T10:15:30 dwc
```

Create or update a file called **nick**; the resulting file has both the last data modification and last data access timestamps set to November 12, 2007 at 10:15:30 UTC:

```
touch -d 2007-11-12T10:15:30Z nick
```

Create or update a file called **gwc**; the resulting file has both the last data modification and last data access timestamps set to November 12, 2007 at 10:15:30 local time with a fractional second timestamp of .002 seconds:

```
touch -d 2007-11-12T10:15:30.002 gwc
```

Create or update a file called **ajosey**; the resulting file has both the last data modification and last data access timestamps set to November 12, 2007 at 10:15:30 UTC with a fractional second timestamp of .002 seconds:

```
touch -d "2007-11-12 10:15:30.002Z" ajosey
```

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #118 is applied, allowing `touch` to support times that precede the Epoch.

Austin Group Interpretation 1003.1-2001 #193 is applied, adding the `-d` option with support for subsecond timestamps.

SD5-XCU-ERN-45 is applied, adding a new paragraph to the RATIONALE noting that if at least two operands are specified and the first operand is an eight or ten-digit decimal integer, the first operand will be taken to be a *file* operand, whereas in previous versions of the standard it would have been taken to be an (obsolescent) *date_time* operand.

Changes are made related to support for finegrained timestamps.

tput

Purpose: Change terminal characteristics.

Synopsis: `tput [-T type] operand...`

Derivation: First released in Issue 4.

Issue 7: The `tput` utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

tr

Purpose: Translate characters.

Synopsis: `tr [-c|-C] [-s] string1 string2`
`tr -s [-c|-C] string1`
`tr -d [-c|-C] string1`
`tr -ds [-c|-C] string1 string2`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #132 is applied, clarifying that the behavior is unspecified if an unescaped trailing <backslash> is present in *string1* or *string2*.

true

Purpose: Return true value.

Synopsis: `true`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

tsort

Purpose: Topological sort.

Synopsis: `tsort [file]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is '-' and the implementation treats the '-' as meaning standard input.

The *tsort* utility is moved from the XSI option to the Base.

tty

Purpose: Return user's terminal name.

Synopsis: `tty`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

type

Purpose: Write a description of command type.

XSI Synopsis: `type name...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

ulimit

Purpose: Set or report file size limit.

XSI Synopsis: `ulimit [-f] [blocks]`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

umask

Purpose: Get or set the file mode creation mask.

Synopsis: `umask [-S] [mask]`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

unalias

Purpose: Remove alias definitions.

Synopsis: `unalias alias-name...`
`unalias -a`

Derivation: First released in Issue 4.

Issue 7: The *unalias* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

uname

Purpose: Return system name.

Synopsis: `uname [-amnrsv]`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

uncompress

Purpose: Expand compressed data.

XSI Synopsis: `uncompress [-cfv] [file...]`

Derivation: First released in Issue 4.

Issue 7: SD5-XCU-ERN-26 is applied, clarifying that this utility is allowed to break the Utility Syntax Guidelines by having ten letters in its name.

unexpand

Purpose: Convert spaces to tabs.

Synopsis: `unexpand [-a|-t tablist] [file...]`

Derivation: First released in Issue 4.

Issue 7: The *unexpand* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

unget

Purpose: Undo a previous get of an SCCS file (**DEVELOPMENT**).

XSI Synopsis: `unget [-ns] [-r SID] file...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

uniq

Purpose: Report or filter out repeated lines in a file.

Synopsis: `uniq [-c|-d|-u] [-f fields] [-s char] [input_file [output_file]]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized as an option delimiter in the OPTIONS section.

Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDOUT section to reflect that standard output is also used if an *output_file* operand is '-' and the implementation treats the '-' as meaning standard output.

Austin Group Interpretation 1003.1-2001 #133 is applied, clarifying that the trailing <newline> of each line in the input is ignored when doing comparisons.

unlink

Purpose: Call the *unlink()* function.

XSI Synopsis: `unlink file`

Derivation: First released in Issue 5.

Issue 7: No functional changes are made in this issue.

uucp

Purpose: System-to-system copy.

UU Synopsis: `uucp [-cCdfjmr] [-n user] source-file... destination-file`

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-46 is applied, moving this utility to the UUCP Utilities Option Group.

uudecode

Purpose: Decode a binary file.

Synopsis: `uudecode [-o outfile] [file]`

Derivation: First released in Issue 4.

Issue 7: The *uudecode* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

uuencode

Purpose: Encode a binary file.

Synopsis: `uuencode [-m] [file] decode_pathname`

Derivation: First released in Issue 4.

Issue 7: The *uuencode* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

uustat

Purpose: uucp status enquiry and job control.

UU Synopsis: `uustat [-q|-k jobid|-r jobid]`
`uustat [-s system] [-u user]`

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-46 is applied, moving this utility to the UUCP Utilities Option Group.

uux

Purpose: Remote command execution.

UU Synopsis: `uux [-jnp] command-string`

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-46 is applied, moving this utility to the UUCP Utilities Option Group.

val

Purpose: Validate SCCS files (**DEVELOPMENT**).

XSI Synopsis: `val -`
`val [-s] [-m name] [-r SID] [-y type] file...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

vi

Purpose: Screen-oriented (visual) display editor.

UP Synopsis: `vi [-rR] [-c command] [-t tagstring] [-w size] [file...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized as an option delimiter in the OPTIONS section.

Austin Group Interpretation 1003.1-2001 #087 is applied, updating the Put from Buffer Before (**P**) command description to address multi-line requirements.

wait

Purpose: Await process completion.

Synopsis: `wait [pid...]`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

wc

Purpose: Word, line, and byte or character count.

Synopsis: `wc [-c|-m] [-lw] [file...]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #092 is applied, changing the STDIN section to reflect that standard input is also used if a *file* operand is '-' and the implementation treats the '-' as meaning standard input.

what

Purpose: Identify SCCS files (**DEVELOPMENT**).

XSI Synopsis: `what [-s] file...`

Derivation: First released in Issue 2.

Issue 7: No functional changes are made in this issue.

who

Purpose: Display who is on the system.

XSI Synopsis: `who [-mTu] [-abdHlprt] [file]`

XSI `who [-mu] -s [-bHlprt] [file]`

`who -q [file]`

`who am i`

`who am I`

Derivation: First released in Issue 2.

Issue 7: SD5-XCU-ERN-58 is applied, clarifying the **-b** option.

The *who* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

write

Purpose: Write to another user.

Synopsis: `write user_name [terminal]`

Derivation: First released in Issue 2.

Issue 7: The *write* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

xargs

Purpose: Construct argument lists and invoke utility.

XSI Synopsis: `xargs [-ptx] [-E eofstr] [-I replstr|-L number|-n number]
[-s size] [utility [argument...]]`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #123 is applied, changing the description of the *xargs* **-I** option.

SD5-XCU-ERN-68 is applied, changing requirements related to the **-x** option and changing the SYNOPSIS to show that the **-I**, **-L** and **-n** options are mutually exclusive.

SD5-XCU-ERN-128 is applied, clarifying the DESCRIPTION of the logical end-of-file string.

SD5-XCU-ERN-132 is applied, updating the EXAMPLES section to demonstrate how to quote *xargs* input appropriately, and the use of **-E ""** to prevent accidental logical end-of-file processing.

yacc

Purpose: Yet another compiler compiler (**DEVELOPMENT**).

CD Synopsis: `yacc [-dltv] [-b file_prefix] [-p sym_prefix] grammar`

Derivation: First released in Issue 2.

Issue 7: Austin Group Interpretation 1003.1-2001 #190 is applied, clarifying the requirements for generated code to conform to the ISO C standard.

Austin Group Interpretation 1003.1-2001 #191 is applied, clarifying the handling of C-language trigraphs and curly brace preprocessing tokens.

SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply (options can be interspersed with operands).

zcat

Purpose: Expand and concatenate data.

XSI Synopsis: `zcat [file...]`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

Headers Migration

13.1 Introduction

This chapter contains a section for each header defined in XBD, Issue 7. Each section contains the SYNOPSIS, gives the derivation of the header, and identifies syntax and semantic changes made to the header in Issue 7 (if any). Only changes that might affect an application programmer are identified.

13.2 Headers

<aio.h>

Purpose: Asynchronous input and output.

Synopsis: `#include <aio.h>`

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: The **<aio.h>** header is moved from the Asynchronous Input and Output option to the Base.

This reference page is clarified with respect to macros and symbolic constants, and type and structure definitions from other headers are added.

<arpa/inet.h>

Purpose: Definitions for Internet operations.

Synopsis: `#include <arpa/inet.h>`

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: No functional changes are made in this issue.

<assert.h>

Purpose: Verify program assertion.
 Synopsis: `#include <assert.h>`
 Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.
 Issue 7: No functional changes are made in this issue.

<complex.h>

Purpose: Complex arithmetic.
 Synopsis: `#include <complex.h>`
 Derivation: First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.
 Issue 7: No functional changes are made in this issue.

<cpio.h>

Purpose: Cpio archive values.
 Synopsis: `#include <cpio.h>`
 Derivation: First released in the Headers Interface, Issue 3 specification. Derived from the POSIX.1-1988 standard.
 Issue 7: The **<cpio.h>** header is moved from the XSI option to the Base.
 This reference page is clarified with respect to macros and symbolic constants.

<ctype.h>

Purpose: Character types.
 Synopsis: `#include <ctype.h>`
 Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.
 Issue 7: The `*_l()` functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

<dirent.h>

Purpose: Format of directory entries.
 Synopsis: `#include <dirent.h>`
 Derivation: First released in Issue 2.
 Issue 7: The `alphasort()`, `dirfd()`, and `scandir()` functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 1.
 The `fdopendir()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.
 Austin Group Interpretation 1003.1-2001 #110 is applied, clarifying that the **DIR** type may be defined as an incomplete type.

<dlfcn.h>

Purpose: Dynamic linking.

Synopsis: `#include <dlfcn.h>`

Derivation: First released in Issue 5.

Issue 7: The **<dlfcn.h>** header is moved from the XSI option to the Base.
This reference page is clarified with respect to macros and symbolic constants.

<errno.h>

Purpose: System error numbers.

Synopsis: `#include <errno.h>`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #050 is applied, allowing [ENOTSUP] and [EOPNOTSUPP] to be the same values.

The [ENOTRECOVERABLE] and [EOWNERDEAD] errors are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Functionality relating to the XSI STREAMS option is marked obsolescent.

Functionality relating to the Threads option is moved to the Base.

This reference page is clarified with respect to macros and symbolic constants.

<fcntl.h>

Purpose: File control options.

Synopsis: `#include <fcntl.h>`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #144 is applied, adding the O_TTY_INIT flag.

Austin Group Interpretation 1003.1-2001 #171 is applied, adding the F_DUPFD_CLOEXEC and O_CLOEXEC flags.

The *openat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The O_EXEC and O_SEARCH flags are added.

Additional flags are added to support *faccessat()*, *fchmodat()*, *fchownat()*, *fstatat()*, *linkat()*, *openat()*, and *unlinkat()*.

This reference page is clarified with respect to macros and symbolic constants.

Changes are made related to support for finegrained timestamps.

<fenv.h>

Purpose: Floating-point environment.

Synopsis: `#include <fenv.h>`

Derivation: First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

Issue 7: ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #37 (SD5-XBD-ERN-49) is applied, clarifying that if no floating-point exception macros are defined by the implementation, `FE_ALL_EXCEPT` shall be defined as zero.

ISO/IEC 9899:1999 standard, Technical Corrigendum 3 #36 is applied, requiring that the floating-point exception macros expand to integer constant expressions with values that are bitwise-distinct.

SD5-XBD-ERN-48 and SD5-XBD-ERN-69 are applied, clarifying that implementations which support the IEC 60559 Floating-Point option are required to define all five floating-point exception macros and all four rounding direction macros.

This reference page is clarified with respect to macros and symbolic constants.

<float.h>

Purpose: Floating types.

Synopsis: `#include <float.h>`

Derivation: First released in Issue 4. Derived from the ISO C standard.

Issue 7: ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #4 (SD5-XBD-ERN-50) is applied, clarifying that an implementation may give zero and non-numeric values, such as infinities and NaNs, a sign, or may leave them unsigned.

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #5 (SD5-XBD-ERN-51) is applied, extending the text concerning floating-point accuracy to cover conversion between floating-point internal representations and string representations performed by the functions in **<stdio.h>**, **<stdlib.h>**, and **<wchar.h>**.

<fmtmsg.h>

Purpose: Message display structures.

XSI Synopsis: `#include <fmtmsg.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

<fnmatch.h>

Purpose: Filename-matching types.

Synopsis: `#include <fnmatch.h>`

Derivation: First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 7: The obsolescent `FNM_NOSYS` constant is removed.

This reference page is clarified with respect to macros and symbolic constants.

<ftw.h>

Purpose: File tree traversal.

XSI Synopsis: `#include <ftw.h>`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The *ftw()* function is marked obsolescent.

This reference page is clarified with respect to macros and symbolic constants.

<glob.h>

Purpose: Pathname pattern-matching types.

Synopsis: `#include <glob.h>`

Derivation: First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 7: SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

The obsolescent GLOB_NOSYS constant is removed.

This reference page is clarified with respect to macros and symbolic constants.

<grp.h>

Purpose: Group structure.

Synopsis: `#include <grp.h>`

Derivation: First released in Issue 1.

Issue 7: SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

<iconv.h>

Purpose: Codeset conversion facility.

Synopsis: `#include <iconv.h>`

Derivation: First released in Issue 4.

Issue 7: SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

The **<iconv.h>** header is moved from the XSI option to the Base.

<inttypes.h>

Purpose: Fixed size integer types.

Synopsis: `#include <inttypes.h>`

Derivation: First released in Issue 5.

Issue 7: No functional changes are made in this issue.

<iso646.h>

Purpose: Alternative spellings.

Synopsis: `#include <iso646.h>`

Derivation: First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: No functional changes are made in this issue.

<langinfo.h>

Purpose: Language information constants.

Synopsis: `#include <langinfo.h>`

Derivation: First released in Issue 2.

Issue 7: The **<langinfo.h>** header is moved from the XSI option to the Base.

The `nl_langinfo_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

This reference page is clarified with respect to macros and symbolic constants, and a reference to **<locale.h>** for the `locale_t` type is added.

<libgen.h>

Purpose: Definitions for pattern matching functions.

XSI Synopsis: `#include <libgen.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

<limits.h>

Purpose: Implementation-defined constants.

Synopsis: `#include <limits.h>`

Derivation: First released in Issue 1.

Issue 7: Austin Group Interpretation 1003.1-2001 #143 is applied, allowing implementations to support pathnames longer than `{PATH_MAX}`.

Austin Group Interpretation 1003.1-2001 #173 is applied, updating the descriptions of `{TRACE_EVENT_NAME_MAX}` and `{TRACE_NAME_MAX}` to not include the terminating null.

SD5-XBD-ERN-36 is applied, changing the description of `{RE_DUP_MAX}` to clarify that it applies to both BREs and EREs.

`{NL_NMAX}` is removed; it should have been removed in Issue 6.

The Trace option values are marked obsolescent.

The `{ATEXIT_MAX}`, `{LONG_BIT}`, `{NL_MSGMAX}`, `{NL_SETMAX}`, `{NL_TEXTMAX}`, and `{WORD_BIT}` values are moved from the XSI option to the Base.

Functionality relating to the Asynchronous Input and Output, Realtime Signals Extension, Threads, and Timers options is moved to the Base.

This reference page is clarified with respect to macros and symbolic constants.

<locale.h>

Purpose: Category macros.

Synopsis: `#include <locale.h>`

Derivation: First released in Issue 3.

Included for alignment with the ISO C standard.

Issue 7: The *duplocale()*, *freelocale()*, *newlocale()*, and *uselocale()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

This reference page is clarified with respect to macros and symbolic constants.

<math.h>

Purpose: Mathematical declarations.

Synopsis: `#include <math.h>`

Derivation: First released in Issue 1.

Issue 7: ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #47 (SD5-XBD-ERN-52) is applied, updating the wording of the `FP_FAST_FMA` macro to require that it expands to the integer constant 1 if it is defined.

The `MAXFLOAT` constant is marked obsolescent. Applications should use `FLT_MAX` as described in the **<float.h>** header instead.

This reference page is clarified with respect to macros and symbolic constants.

<monetary.h>

Purpose: Monetary types.

Synopsis: `#include <monetary.h>`

Derivation: First released in Issue 4.

Issue 7: The **<monetary.h>** header is moved from the XSI option to the Base.

The *strmon_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

A reference to **<locale.h>** for the `locale_t` type is added.

<mqueue.h>

Purpose: Message queues (**REALTIME**).

MSG Synopsis: `#include <mqueue.h>`

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: Type and structure definitions from other headers are added.

<ndbm.h>

Purpose: Definitions for ndbm database operations.

XS1 Synopsis: `#include <ndbm.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

<netdb.h>

Purpose: Definitions for network database operations.

Synopsis: `#include <netdb.h>`

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: SD5-XBD-ERN-14 is applied, changing the description of the *s_port* member of the **servent** structure to clarify the way in which port numbers are converted to and from network byte order.

The obsolescent *h_errno* external integer, and the obsolescent *gethostbyaddr()* and *gethostbyname()* functions are removed, along with the `HOST_NOT_FOUND`, `NO_DATA`, `NO_RECOVERY`, and `TRY_AGAIN` macros.

This reference page is clarified with respect to macros and symbolic constants.

<net/if.h>

Purpose: Sockets local interfaces.

Synopsis: `#include <net/if.h>`

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

<netinet/in.h>

Purpose: Internet address family.

Synopsis: `#include <netinet/in.h>`

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

<netinet/tcp.h>

Purpose: Definitions for the Internet Transmission Control Protocol (TCP).

Synopsis: `#include <netinet/tcp.h>`

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

<nl_types.h>

Purpose: Data types.

Synopsis: `#include <nl_types.h>`

Derivation: First released in Issue 2.

Issue 7: The **<nl_types.h>** header is moved from the XSI option to the Base.

This reference page is clarified with respect to macros and symbolic constants.

<poll.h>

Purpose: Definitions for the *poll()* function.

Synopsis: `#include <poll.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: The **<poll.h>** header is moved from the XSI option to the Base.

<pthread.h>

Purpose: Threads.

Synopsis: `#include <pthread.h>`

Derivation: First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 7: SD5-XBD-ERN-55 is applied, adding the **restrict** keyword to the *pthread_mutex_timedlock()* function prototype so that it matches the definition in XSH.

Austin Group Interpretation 1003.1-2001 #048 is applied, reinstating the PTHREAD_RWLOCK_INITIALIZER symbol.

The **<pthread.h>** header is moved from the Threads option to the Base.

The PTHREAD_MUTEX_NORMAL, PTHREAD_MUTEX_ERRORCHECK, PTHREAD_MUTEX_RECURSIVE, and PTHREAD_MUTEX_DEFAULT extended mutex types are moved from the XSI option to the Base.

The PTHREAD_MUTEX_ROBUST and PTHREAD_MUTEX_STALLED symbols and the *pthread_mutex_consistent()*, *pthread_mutexattr_getrobust()*, and *pthread_mutexattr_setrobust()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Functionality relating to the Thread Priority Protection and Thread Priority Inheritance options is changed to be Non-Robust Mutex or Robust Mutex Priority Protection and Non-Robust Mutex or Robust Mutex Priority Inheritance, respectively.

This reference page is clarified with respect to macros and symbolic constants.

<pwd.h>

Purpose: Password structure.

Synopsis: `#include <pwd.h>`

Derivation: First released in Issue 1.

Issue 7: SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

<regex.h>

Purpose: Regular expression matching types.

Synopsis: `#include <regex.h>`

Derivation: First released in Issue 4.

Originally derived from the ISO POSIX-2 standard.

Issue 7: SD5-XBD-ERN-60 is applied, removing the requirement that the type **regoff_t** can hold the largest value that can be stored in type **off_t**, and adding the requirement that the type **regoff_t** can hold the largest value that can be stored in type **ptrdiff_t**.

The obsolescent REG_ENOSYS constant is removed.

This reference page is clarified with respect to macros and symbolic constants.

<sched.h>

Purpose: Execution scheduling.

Synopsis: `#include <sched.h>`

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: Austin Group Interpretation 1003.1-2001 #064 is applied, correcting the option markings.

The **<sched.h>** header is moved from the Threads option to the Base.

Definitions for the **pid_t** and **time_t** types and the **timespec** structure are added.

<search.h>

Purpose: Search tables.

XSI Synopsis: `#include <search.h>`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

<semaphore.h>

Purpose: Semaphores.

Synopsis: `#include <semaphore.h>`

Derivation: First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 7: SD5-XBD-ERN-57 is applied, allowing the header to make visible symbols from the **<time.h>** header.

The **<semaphore.h>** header is moved from the Semaphores option to the Base.

This reference page is clarified with respect to macros and symbolic constants.

<setjmp.h>

Purpose: Stack environment declarations.

Synopsis: `#include <setjmp.h>`

Derivation: First released in Issue 1.

Issue 7: No functional changes are made in this issue.

<signal.h>

Purpose: Signals.

Synopsis: `#include <signal.h>`

Derivation: First released in Issue 1.

Issue 7: SD5-XBD-ERN-39 is applied, removing the **sigstack** structure which should have been removed at the same time as the LEGACY `sigstack()` function.

SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

Austin Group Interpretation 1003.1-2001 #034 is applied, moving SIGPOLL from the XSI option to the XSI STREAMS option.

The `psiginfo()` and `psignal()` functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

Functionality relating to the XSI STREAMS option is marked obsolescent.

The SA_RESETHAND, SA_RESTART, SA_NOCLDWAIT, and SA_NODEFER constants are moved from the XSI option to the Base.

Functionality relating to the Realtime Signals Extension option is moved to the Base.

This reference page is clarified with respect to macros and symbolic constants, and type and structure definitions from other headers are added.

The descriptions of SIGRTMIN and SIGRTMAX are updated to clarify that they expand to positive integer expressions with type **int**, but which need not be constant expressions.

The APPLICATION USAGE section is updated to describe the conditions under which the `si_pid` and `si_uid` members of **siginfo_t** are required to be valid.

<spawn.h>

Purpose: Spawn (**ADVANCED REALTIME**).

SPN

Synopsis: `#include <spawn.h>`

Derivation: First released in Issue 6. Included for alignment with IEEE Std 1003.1d-1999.

Issue 7: This reference page is clarified with respect to macros and symbolic constants, and type and structure definitions from other headers are added.

<stdarg.h>

Purpose: Handle variable argument list.

Synopsis: `#include <stdarg.h>`

Derivation: First released in Issue 4. Derived from the ANSI C standard.

Issue 7: No functional changes are made in this issue.

<stdbool.h>

Purpose: Boolean type and values.

Synopsis: `#include <stdbool.h>`

Derivation: First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

Issue 7: No functional changes are made in this issue.

<stddef.h>

Purpose: Standard type definitions.

Synopsis: `#include <stddef.h>`

Derivation: First released in Issue 4. Derived from the ANSI C standard.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

SD5-XBD-ERN-53 is applied, updating the definition of **wchar_t** to align with ISO/IEC 9899:1999 standard, Technical Corrigendum 3 in relation to the `__STDC_MB_MIGHT_NEQ_WC__` indicator macro.

<stdint.h>

Purpose: Integer types.

Synopsis: `#include <stdint.h>`

Derivation: First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

Issue 7: ISO/IEC 9899:1999 standard, Technical Corrigendum 3 #40 is applied, requiring the argument to the `INT*_C()` macros to be an unsuffixed integer constant.

SD5-XBD-ERN-67 is applied, updating the RATIONALE to clarify that `{SCHAR_MIN}` has the value `-128`.

<stdio.h>

Purpose: Standard buffered input/output.

Synopsis: `#include <stdio.h>`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #172 is applied, adding rationale about a conflict for the definition of {TMP_MAX} with the ISO C standard and the related ISO C defect report.

SD5-XBD-ERN-99 is applied, adding APPLICATION USAGE about {FOPEN_MAX} and the use of file descriptors not associated with streams.

The *dprintf()*, *fmemopen()*, *getdelim()*, *getline()*, *open_memstream()*, and *vdprintf()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The *renameat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The *gets()*, *tmpnam()*, and *tempnam()* functions and the `L_tmpnam` macro are marked obsolescent.

This reference page is clarified with respect to macros and symbolic constants, and a reference to **<sys/types.h>** for the **off_t** type is added.

<stdlib.h>

Purpose: Standard library definitions.

Synopsis: `#include <stdlib.h>`

Derivation: First released in Issue 3.

Issue 7: The LEGACY functions are removed.

The *mkdtemp()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The *rand_r()* function is marked obsolescent.

This reference page is clarified with respect to macros and symbolic constants.

The type of the first argument to *setstate()* is changed from **const char *** to **char ***.

<string.h>

Purpose: String operations.

Synopsis: `#include <string.h>`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: SD5-XBD-ERN-15 is applied, correcting the prototype for the *strerror_r()* function.

The *stpncpy()*, *stpncpy()*, *strndup()*, *strnlen()*, and *strsignal()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The *strcoll_l()*, *strerror_l()*, and *strxfrm_l()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

This reference page is clarified with respect to macros and symbolic constants, and a reference to **<locale.h>** for the **locale_t** type is added.

<strings.h>

Purpose: String operations.

Synopsis: `#include <strings.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

The LEGACY functions are removed.

The **<strings.h>** header is moved from the XSI option to the Base.

The *strcasecmp_1()* and *strncasecmp_1()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

A reference to **<locale.h>** for the **locale_t** type is added.

<stropts.h>

Purpose: STREAMS interface (**STREAMS**).

OB XSR Synopsis: `#include <stropts.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: SD5-XBD-ERN-87 is applied, correcting an error in the **strecvfd** structure.

The **<stropts.h>** header is marked obsolescent.

This reference page is clarified with respect to macros and symbolic constants.

<sys/ipc.h>

Purpose: XSI interprocess communication access structure.

XSI Synopsis: `#include <sys/ipc.h>`

Derivation: First released in Issue 2. Derived from System V Release 2.0.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

<syslog.h>

Purpose: Definitions for system error logging.

XSI Synopsis: `#include <syslog.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

<sys/mman.h>

Purpose: Memory management declarations.

Synopsis: `#include <sys/mman.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: Functionality relating to the Memory Protection and Memory Mapped Files options is moved to the Base.

This reference page is clarified with respect to macros and symbolic constants.

<sys/msg.h>

Purpose: XSI message queue structures.

XSI Synopsis: `#include <sys/msg.h>`

Derivation: First released in Issue 2. Derived from System V Release 2.0.

Issue 7: Austin Group Interpretation 1003.1-2001 #179 is applied, clarifying that everything from **<sys/ipc.h>** is made visible by **<sys/msg.h>**.

This reference page is clarified with respect to macros and symbolic constants.

<sys/resource.h>

Purpose: Definitions for XSI resource operations.

XSI Synopsis: `#include <sys/resource.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

<sys/select.h>

Purpose: Select types.

Synopsis: `#include <sys/select.h>`

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

<sys/sem.h>

Purpose: XSI semaphore facility.

XSI Synopsis: `#include <sys/sem.h>`

Derivation: First released in Issue 2. Derived from System V Release 2.0.

Issue 7: Austin Group Interpretation 1003.1-2001 #179 is applied, clarifying that everything from **<sys/ipc.h>** is made visible by **<sys/sem.h>**.

This reference page is clarified with respect to macros and symbolic constants.

<sys/shm.h>

Purpose: XSI shared memory facility.

XSI Synopsis: `#include <sys/shm.h>`

Derivation: First released in Issue 2. Derived from System V Release 2.0.

Issue 7: Austin Group Interpretation 1003.1-2001 #179 is applied, clarifying that everything from **<sys/ipc.h>** is made visible by **<sys/shm.h>**.

This reference page is clarified with respect to macros and symbolic constants.

<sys/socket.h>

Purpose: Main sockets header.

Synopsis: `#include <sys/socket.h>`

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **ssize_t** type.

The `MSG_NOSIGNAL` symbolic constant is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

This reference page is clarified with respect to macros and symbolic constants, and a reference to **<sys/types.h>** for the **size_t** type is added.

<sys/stat.h>

Purpose: Data returned by the `stat()` function.

Synopsis: `#include <sys/stat.h>`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: SD5-XSH-ERN-161 is applied, updating the DESCRIPTION to clarify that the descriptions of the interfaces should be consulted in order to determine which structure members have meaningful values.

The `fchmodat()`, `fstatat()`, `mkdirat()`, `mkfifoat()`, `mknodat()`, and `utimensat()` functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

This reference page is clarified with respect to macros and symbolic constants.

Changes are made related to support for finegrained timestamps, and the `futimens()` function and the `UTIME_NOW` and `UTIME_OMIT` symbolic constants are added.

<sys/statvfs.h>

Purpose: VFS File System information structure.

Synopsis: `#include <sys/statvfs.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: The **<sys/statvfs.h>** header is moved from the XSI option to the Base.

This reference page is clarified with respect to macros and symbolic constants.

<sys/time.h>

Purpose: Time types.

XSI Synopsis: `#include <sys/time.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

<sys/times.h>

Purpose: File access and modification times structure.

Synopsis: `#include <sys/times.h>`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

<sys/types.h>

Purpose: Data types.

Synopsis: `#include <sys/types.h>`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #033 is applied, requiring **key_t** to be an arithmetic type.

The Trace option types are marked obsolescent.

The **clock_t** and **id_t** types are moved from the XSI option to the Base.

Functionality relating to the Barriers, Spin Locks, Timers, and Threads options is moved to the Base.

<sys/uio.h>

Purpose: Definitions for vector I/O operations.

XSI Synopsis: `#include <sys/uio.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

<sys/un.h>

Purpose: Definitions for UNIX domain sockets.

Synopsis: `#include <sys/un.h>`

Derivation: First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7: The value for `{_POSIX_PATH_MAX}` stated in APPLICATION USAGE is updated to 256.

<sys/utsname.h>

Purpose: System name structure.

Synopsis: `#include <sys/utsname.h>`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: No functional changes are made in this issue.

<sys/wait.h>

Purpose: Declarations for waiting.

Synopsis: `#include <sys/wait.h>`

Derivation: First released in Issue 3.

Included for alignment with the POSIX.1-1988 standard.

Issue 7: The `waitid()` function and symbolic constants for its `options` argument are moved to the Base.

The description of the `WNOHANG` constant is clarified.

<tar.h>

Purpose: Extended tar definitions.

Synopsis: `#include <tar.h>`

Derivation: First released in Issue 3. Derived from the POSIX.1-1988 standard.

Issue 7: This reference page is clarified with respect to macros and symbolic constants.

<termios.h>

Purpose: Define values for termios.

Synopsis: `#include <termios.h>`

Derivation: First released in Issue 3.

Included for alignment with the ISO POSIX-1 standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #144 is applied, moving functionality relating to the `IXANY` symbol from the `XSI` option to the Base.

This reference page is clarified with respect to macros and symbolic constants, and a reference to **<sys/types.h>** for the `pid_t` type is added.

<tgmath.h>

Purpose: Type-generic macros.

Synopsis: `#include <tgmath.h>`

Derivation: First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

Issue 7: Austin Group Interpretation 1003.1-2001 #184 is applied, clarifying the functions for which a corresponding type-generic macro exists with the same name as the function.

<time.h>

Purpose: Time types.

Synopsis: `#include <time.h>`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: The `strftime_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

Functionality relating to the Timers option is moved to the Base.

This reference page is clarified with respect to macros and symbolic constants, and type and structure definitions from other headers are added.

The description of `getdate_err` is expanded to state that it is unspecified whether `getdate_err` is a macro or an identifier declared with external linkage, and whether or not it is a modifiable lvalue.

<trace.h>

Purpose: Tracing.

OB TRC Synopsis: `#include <trace.h>`

Derivation: First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7: SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

The **<trace.h>** header is marked obsolescent.

This reference page is clarified with respect to macros and symbolic constants.

<ulimit.h>

Purpose: Ulimit commands.

OB XSI Synopsis: `#include <ulimit.h>`

Derivation: First released in Issue 3.

Issue 7: The **<ulimit.h>** header is marked obsolescent.

<unistd.h>

Purpose: Standard symbolic constants and types.

Synopsis: `#include <unistd.h>`

Derivation: First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 7: Austin Group Interpretation 1003.1-2001 #026 is applied, clarifying the meanings of the values `-1`, `0`, and greater than `0` for constants for Options and Option Groups, and making an undefined constant mean the same as the value `-1`.

Austin Group Interpretation 1003.1-2001 #047 is applied, adding the `_CS_V7_ENV` constant.

Austin Group Interpretation 1003.1-2001 #166 is applied to permit an additional compiler flag to enable threads.

Austin Group Interpretation 1003.1-2001 #178 is applied, clarifying the values allowed for `_POSIX2_CHAR_TERM`.

SD5-XBD-ERN-41 is applied, adding the `_POSIX2_SYMLINKS` constant.

SD5-XBD-ERN-77 is applied, moving `_POSIX_VDISABLE` out of Constants for Options and Option Groups, since its value does not follow the convention for those constants.

Symbols to support the UUCP Utilities option are added.

The variables for the supported programming environments are updated to be V7.

The LEGACY and obsolescent symbols are removed.

The *faccessat()*, *fchownat()*, *fexecve()*, *linkat()*, *readlinkat()*, *symlinkat()*, and *unlinkat()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The `_POSIX_TRACE*` constants from the Trace option are marked obsolescent.

The `_POSIX2_PBS*` constants from the Batch Environment Services and Utilities option are marked obsolescent.

Functionality relating to the Asynchronous Input and Output, Barriers, Clock Selection, Memory Mapped Files, Memory Protection, Realtime Signals Extension, Semaphores, Spin Locks, Threads, Timeouts, and Timers options is moved to the Base.

Functionality relating to the Thread Priority Protection and Thread Priority Inheritance options is changed to be Non-Robust Mutex or Robust Mutex Priority Protection and Non-Robust Mutex or Robust Mutex Priority Inheritance, respectively.

The following symbolic constants are added:

```
_SC_THREAD_ROBUST_PRIO_INHERIT
_SC_THREAD_ROBUST_PRIO_PROTECT
```

This reference page is clarified with respect to macros and symbolic constants.

Changes are made related to support for finegrained timestamps.

<utime.h>

Purpose: Access and modification times structure.

OB

Synopsis: `#include <utime.h>`

Derivation: First released in Issue 3.

Issue 7: The **<utime.h>** header is marked obsolescent.

<utmpx.h>

Purpose: User accounting database definitions.

XSI

Synopsis: `#include <utmpx.h>`

Derivation: First released in Issue 4, Version 2.

Issue 7: No functional changes are made in this issue.

<wchar.h>

Purpose: Wide-character handling.

Synopsis: `#include <wchar.h>`

Derivation: First released in Issue 4.

Issue 7: No functional changes are made in this issue.

<wctype.h>

Purpose: Wide-character classification and mapping utilities.

Synopsis: `#include <wctype.h>`

Derivation: First released in Issue 5. Derived from the ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 7: The `*_/()` functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

This reference page is clarified with respect to macros and symbolic constants.

<wordexp.h>

Purpose: Word-expansion types.

Synopsis: `#include <wordexp.h>`

Derivation: First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 7: The obsolescent `WRDE_NOSYS` constant is removed.

This reference page is clarified with respect to macros and symbolic constants.

ISO C Migration

This chapter is by Finnbar P. Murphy. At the time of writing, Finnbar was a software engineer in the Business Critical Systems Group (BCSG) at Compaq Computer Corporation in Nashua, New Hampshire.

14.1 Introduction

The original ISO/IEC C language programming standard (the ISO/IEC 9899:1990 standard) was adopted by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) in 1990. Subsequently, two technical corrigenda (TC1 and TC2) were approved together with the normative ISO/IEC 9899:1990/Amendment 1:1995 (E), Multibyte Support Extension.²

At the end of 1993, there was general agreement that work should start on the next revision of the standard. The revised standard (C99) was sent for FCD ballot in August 1998, and adopted by ISO/IEC in 1999 as the ISO/IEC 9899:1999 standard.

This chapter is intended to provide the reader with a good, but not exhaustive, overview of the differences between the two revisions of the standard. Thus the reader is strongly advised to reference the ISO/IEC 9899:1999 standard for specific details.

14.2 Language Changes

A significant number of changes occurred in the standard, including new keywords and types, type qualifiers, better floating-point support, and support for complex numbers.

14.2.1 New Keywords

The following new keywords were defined:

- **inline**
- **restrict**
- **_Bool**
- **_Complex**

2. Information about the ISO C Working Group (JTC1/SC22/WG14) can be found at: wwwold.dkuug.dk/JTC1/SC22/WG14/.

- **_Imaginary**
- **long long**

14.2.2 New Types

Two new types were added:

- **_Bool**
- **long long**

The **long long** type is an integer type with at least 64 bits of precision.

Note: In some programming models such as LP64 and ILP64, **long long** and **long** are equivalent. In the others—for example, LLP64—**long long** is larger than **long**.

14.2.3 Type Qualifiers

Type qualifiers are now idempotent. If a type qualifier appears more than once (either directly or indirectly) in a type specification, it is as if it appeared only once. Thus **const const int fpm**; and **const int fpm**; are equivalent.

restrict is a new type qualifier which enables programs to be written so that compilers can produce significantly faster executables. It is intended to be used only with pointers. Objects referenced through a **restrict**-qualified pointer are special in that all references to the object must directly or indirectly use the value of the **restrict**-qualified pointer. It is intended to facilitate better alias analysis by compilers. In the absence of this qualifier, other pointers can alias the object and prevent compiler optimizations since a compiler may not be able to determine that different pointers are being used to reference different objects. Note that a restricted pointer and a non-restricted pointer can be aliases.

A number of function definitions were modified to take advantage of the **restrict** qualifier. A typical example is the *fopen()* function which was changed from:

```
FILE *fopen(const char *filename, const char *mode);
```

to:

```
FILE *fopen(const char *restrict filename,
            const char *restrict mode);
```

Changed functions include:

<i>fgetpos()</i>	<i>freopen()</i>	<i>memcpy()</i>	<i>strncpy()</i>	<i>vwprintf()</i>
<i>fgets()</i>	<i>fwprintf()</i>	<i>setbuf()</i>	<i>strxfrm()</i>	<i>wcstod()</i>
<i>fgetws()</i>	<i>fwrite()</i>	<i>setvbuf()</i>	<i>swprintf()</i>	<i>wcstol()</i>
<i>fopen()</i>	<i>fwscanf()</i>	<i>strcat()</i>	<i>swscanf()</i>	<i>wcstombs()</i>
<i>fputs()</i>	<i>mbstowcs()</i>	<i>strcpy()</i>	<i>vfwprintf()</i>	<i>wcstoul()</i>
<i>fread()</i>	<i>mbtowc()</i>	<i>strncat()</i>	<i>vswprintf()</i>	<i>wprintf()</i>

14.2.4 Boolean

The standard now supports a boolean type `_Bool` which is an integer type which can hold either 0 or 1.

The header `<stdbool.h>` also defines the macro `bool` which expands to `_Bool`, `true` which expands to the integer constant 1, and `false` which expands to the integer constant 0.

14.2.5 Universal Character Names

Prior to this revision of the standard, “native” characters, in the form of multibyte and wide characters, could be used in string literals and character constants, but not as part of an identifier.

This standard introduced the concept of a universal character name (UCN) that may be used in identifiers, character constants, and string literals to designate characters that are not in the basic character set.

The two forms of a UCN are:

```
\unnnn      where nnnn is hex-quad
\Unnnnnnnn  where nnnnnnnn is hex-quad hex-quad
```

A *hex-quad* consists of 4 hexadecimal digits.

The UNC `\Unnnnnnnn` designates the character whose eight-digit short identifier as specified by the ISO/IEC 10646-1:2000 standard is `nnnnnnnn`.

Similarly, the UCN `\unnnn` can be used to designate a given character whose four-digit short identifier as specified by the ISO/IEC 10646-1:2000 standard is `nnnn` (and whose eight-digit short identifier is `0000nnnn`).

There are a number of disallowed characters; that is, those in the basic character set, and code positions reserved in the ISO/IEC 10646-1:2000 standard for control and DELETE characters and UTF-16.

Note: A strictly conforming program may use only the extended characters listed in Annex I (Universal Character Names for Identifiers) and may not begin an identifier with an extended digit. Also, use of native characters in comments has always been strictly conforming, though what happens when such a program is printed in a different locale is unspecified.

14.2.6 inline

The **inline** keyword is intended to provide users with a portable way to suggest to implementations that inlining a function might result in program optimizations.

It is a function-specifier that can be used only in function declarations. It was adopted from C++ but extended in such a way that it can be implemented with existing linker technology. The translation unit that contains the definition of an inline function is the unit that provides the external definition for the function. If a function is declared inline in one translation unit, it need not be declared inline in every other translation unit.

14.2.7 Predefined Identifiers

Predefined identifiers are variables that have block scope.

The standard defined one predefined identifier `__func__` which is declared implicitly by the compiler as if, immediately following the opening brace of each function definition, the following declaration was included in the source code:

```
static const char __func__[] = "function-name";
```

where *function-name* is the name of the lexically-enclosing function. This enables a function name to be obtained at runtime.

The `assert()` macro now includes the identifier `__func__` in the output to `stderr`:

```
void assert(scalar expression);
```

Note that the parameter type of the `assert()` macro was also changed from `int` to `scalar`.

14.2.8 Compound Literals

Compound literals (also known as anonymous aggregates) provide a mechanism for specifying constants of aggregate or union type. This eliminates the requirement for temporary variables when an aggregate or union value may only be needed once. Compound literals are primary expressions which can also be combined with designated initializers to form an even more convenient aggregate or union constant notation.

Compound literals are created using the notation:

```
( type-name ) { initializer-list }
```

For example:

```
int *ap = (int a[]) {1, 2, 3};
```

Note that a trailing comma before the closing brace is permitted.

14.2.9 Designated Initializers

Designated initializers provide a mechanism for initializing aggregates such as sparse arrays, a common requirement in numerical programming. This mechanism also allows initialization of sparse structures and initialization of unions via any member, regardless of whether or not it is the first member.

Initializers have a named notation for initializing members. For array elements, the element is designated by `[const-expression]`, for **struct** and **union** members by a dot member-name notation.

For example:

```
struct s { int a; int b; };
struct s mystruct = {.b = 2}; // initialize member b
struct {int a[3], b[3]} w[] = { [0].a = {1}, [1].b = 2 };
```

If an initializer is present, any members not explicitly set are zeroed out. Initializers for **auto** aggregates can be non-constant expressions.

14.3 Decimal Integer Constants

The default type of a decimal integer constant is either **int**, **long**, or **long long** (previously **int**, **long**, and **unsigned long**), depending on which type is large enough to hold the value without overflow.

The standard added LL to specify **long long**, and ULL to specify **unsigned long long**.

14.3.1 String Literals

The standard defines a number of macros as expanding into character string literals that are frequently needed as wide strings.

One example is the format specifier macros in **<inttypes.h>**. Rather than specifying two forms of each macro, one character string literal and one wide string literal, the decision was made to define the result of concatenating a character string literal and a wide string literal as a wide string literal.

14.4 Implicit Declarations

Implicit declaration of functions is no longer permitted by the standard. There must be a least one type specifier otherwise a diagnostic is issued. However, after issuing the diagnostic, an implementation may choose to assume an implicit declaration and continue translation in order to support existing source code.

For example, the declaration *fpm()*; was valid in previous revisions of the standard (equivalent to **int fpm();**) but is now invalid.

14.4.1 sizeof

With the addition of variable length arrays, the *sizeof* operator is a constant expression only if the type of the operand is not a variable length array type.

Note: It is still possible to determine the number of elements in a variable length array *vla* with *sizeof(vla)/sizeof(vla[0])*.

14.4.2 Multiplicative Operators

In previous revisions of the standard, division of integers involving negative operands could round upward or downward in an implementation-defined manner. The standard now mandates that, as in Fortran, the result always truncates toward zero.

For example, both of the following truncate towards zero:

```
-22 / 7 = -3
-22 % 7 = -1
```

This was done to facilitate porting of code from Fortran to C.

14.4.3 Enumeration Specifiers

A common extension to many C implementations is to allow a trailing comma after the list of enumeration constants. The standard now permits this.

14.5 Variable Length Array

A new array type, called a variable length array type, was added to the standard. The number of elements specified in the declaration of a variable length array type is not specified by the source code; rather it is a computed value determined at runtime.

Multi-dimensional variable-length arrays are permitted.

Some things cannot be declared as a variable length array type, including:

- File scope identifiers
- Arrays declared using either **static** or **extern** storage class specifiers
- Structure and union members

The rationale behind this new array type was that some standard method to support runtime array sizing was considered crucial for C's acceptance in the numerical computing world. Before this revision of the standard, the size expression was required to be an integer constant expression.

14.5.1 Array Declarations

The **static** storage class specifier and the type-qualifiers **restrict**, **const**, or **volatile** can now be used inside the square brackets of an array type declaration, but only in the outermost array type derivation of a function parameter.

```
int foo(const int a[static 10]);
```

In the above example, the **static** keyword will guarantee that the pointer to the array *a* is not NULL, and points to an object of the appropriate type.

14.5.2 Array Type Compatibility

Array type compatibility was extended so that variable length arrays are compatible with both an array of known constant size and an array with an incomplete type.

14.5.3 Incomplete Array Structure Members

The last member of a structure with more than one member can now be an incomplete array type. This incomplete member is called a flexible array member.

Consider the following example

```
struct s { int n;  
          double d[];  
};  
  
size_t sz = sizeof( struct s );  
struct s *sp = malloc(sz + 10);
```


The structure pointer **sp** behaves as if the structure **s** had been declared as

```
struct s { int n;
          double d[10];
};
```

The size of the structure is equal to the offset of the last element of an otherwise identical structure that replaces the flexible array member with an array of unspecified length. When a `'.'` or a `'->'` operator point to a structure with a flexible array member and the right operand names that member, it behaves as if that member were replaced with the longest array with the same element type that would not make the structure larger than the object being accessed.

The offset of the array remains that of the flexible array member, even if this would differ from that of the replacement array. If this array would have no elements, it behaves as if it had one element. However, behavior is undefined if any attempt is made to access that element or to generate a pointer one past it.

14.5.4 Blocks

A common coding practice is to always use compound statements for every selection and iteration statement to guard against inadvertent problems when changes are made to the source code.

Because this can lead to surprising behavior in connection with certain uses of compound literals, the concept of a block was expanded in this revision of the standard.

As in C++, all selection and iteration statements, and their associated substatements, are now defined to be blocks, even if they are not also compound statements. If compound literals are defined in selection or iteration statements, their lifetimes are limited to the implied enclosing block.

14.5.5 The for Statement

The standard now permits loop counter variables as part of a **for** statement. Such a variable is in a new scope (so it does not affect any other variable of the same name), is destroyed at the end of the loop, and must have **auto** or **register** storage class.

```
for (int i = 0; i < 10; i++)
printf("Loop number: %d\n", i);
```

14.5.6 `errno`

For underflow, *errno* is no longer required to be set to [EDOM] or [ERANGE].

14.6 Comments

Support for `//`-style comments was added due to their utility and widespread existing practice, especially in dual C/C++ translators. This is a quiet change which could cause different semantics between this standard and C89. Consider the following example:

```
a = b /*divisor:*/ f
    + e;
```

According to this standard this is the same as:

```
a = b + e;
```

but in previous revisions of the standard it was the same as:

```
a = b / f + e;
```

14.6.1 Hexadecimal Floating-Point Constants

Because hexadecimal notation more clearly expresses the significance of floating constants, the standard now supports hexadecimal floating-point constants.

The binary-exponent part is required, instead of being optional as it is for decimal notation, to avoid ambiguity resulting from an `'f'` suffix being mistaken as a hexadecimal digit. The exponent indicates the power of 2 by which the significant part is to be scaled.

14.6.2 Predefined Macros

New predefined macros include:

<code>__STDC_VERSION__</code>	Defined to be 199901L to indicate the current revision of the standard.
<code>__STDC_HOSTED__</code>	Defined as 1 if the implementation is hosted; otherwise, 0.

14.6.3 Source File Inclusion

The number of significant characters in header and source file names was raised from six to eight, and digits are now allowed.

14.6.4 Translation-Time Arithmetic

The standard now mandates that translation-time arithmetic be done using `intmax_t` or `uintmax_t`, which must comprise at least 64 bits and must match the execution environment.

Previously, a translator was permitted to evaluate expressions using the `long` integer or `unsigned long` integer arithmetic native to the translation environment.

14.6.5 Minimum Maximum Line Length

The minimum maximum line length was increased from 254 to 4095.

14.6.6 Case-Sensitive Identifiers

All identifiers are now case-sensitive. In previous revisions of the standard, it was implementation-defined whether an implementation ignored the case of external identifiers.

14.6.7 #line Directive

This directive now allows the specification of a line number up to $2^{31}-1$. Previously the limit was 32 767.

14.6.8 Empty Argument Macros

Empty arguments are now explicitly allowed. In previous revisions of the standard, this resulted in undefined behavior. Stringification (`#` operator) of an empty argument yields the empty string, concatenation (`##` operator) of an empty argument with a non-empty argument produces the non-empty argument, and concatenation of two empty arguments produces nothing.

14.6.9 Pragmas

Some **pragma** directives have been standardized. Directives whose first preprocessing token is **STDC** are reserved for standardized directives.

As an alternative syntax for a **pragma** directive, the preprocessing operator `_Pragma` is specified. This has the advantage that it can be used in a macro replacement list.

14.6.10 Translation Limits

A number of the program translation limits were significantly increased.

The number of significant initial characters in an internal identifier or a macro name was increased from 31 to 63.

The number of significant characters in an external identifier has increased from 6 to 31 case-sensitive characters.

Note that each universal character name (UCN) specifying a short identifier of 0000FFFF or less is considered to be 6 characters, while a long UCN counts as 10 characters.

While an implementation is not obliged to remember more than the first 63 characters of an identifier with internal linkage, or the first 31 characters of an identifier with external linkage, the programmer is effectively prohibited from intentionally creating two different identifiers that are the same within the appropriate length.

The minimum maximum limit of cases in a switch statement was increased to 1 023.

14.6.11 Token Pasting

The standard replaced non-digit with identifier-non-digit in the grammar to allow the token pasting operator, **##**, to work as expected with characters which are not part of the basic character set.

14.6.12 Variadic Macros

The standard extended the functionality of the punctuator "... " (ellipsis; denoting a variable number of trailing arguments) to function-like macros. For replacement, the variable arguments (including the separating commas) are “collected” into one single extra argument that can be referenced as `__VA_ARGS__` within the macro’s replacement list.

For example:

```
#define MyLog(...) fprintf(stderr, __VA_ARGS__)

main()
{
    int array_bound = 10;
    int array_index = 11;
    .....
    MyLog("ERROR: Index out of bound: %d %d\n", array_index,
        array_bound);
    .....
}
```

There must be at least one argument to match the ellipsis. This requirement avoids problems that might occur when the trailing arguments are included in a list of arguments to another macro or function.

14.6.13 `va_copy()`

In previous revisions of the standard, it was not possible to backtrack and examine one or more arguments a second time when processing a variable argument list. The only way to do this was to reprocess the variable argument list.

The `va_copy()` macro provides a mechanism for copying the **va_list** object used to represent processing of the arguments. Calling the `va_copy()` macro exactly duplicates the **va_list** object.

Note: A separate call to the `va_end()` macro is required to remove the new **va_list** object.

14.7 Headers

The following new headers were added to the standard:

<complex.h>	Defines a number of macros and functions for use with the three complex arithmetic types defined in the standard.
<fenv.h>	Defines a number of types, macros, and functions that can be used to test, control, and access an implementation's floating-point environment.
<inttypes.h>	Defines a type and a number of macros and functions for manipulating integers; <stdint.h> is a subset of this header.
<stdbool.h>	Defines a number of macros for accessing the new Boolean type _Bool and writing Boolean tests.
<tgmath.h>	Defines a large number of type-generic macros that invoke the correct math function from <math.h> or from <complex.h> depending upon their argument types.

14.8 Integer Types

The purpose of the **<inttypes.h>** header is to provide a set of integer types whose definitions are consistent across platforms. Consistent use of these integer types should greatly increase the portability of source code across platforms.

The header **<stdint.h>** is a subset of **<inttypes.h>** and may be more suitable for use in freestanding environments, which might not support the formatted I/O functions. It declares sets of integer types having specified widths and corresponding macros that specify limits of the declared types and construct suitable constants.

The following categories of integer types are defined:

- Types having exact widths
- Types having at least certain specified widths
- Fastest types having at least certain specified widths
- Types wide enough to hold pointers to objects
- Types having greatest width

14.8.1 Exact-Width Integer Types

The **typedef** name **intN_t** designates a signed integer type with width N bits, no padding bits, and a two's complement representation. The **typedef** name **uintN_t** designates an unsigned integer type with width N .

For example, **int16_t** is an unsigned integer type with a width of exactly 16 bits.

Exact-width types are optional. However, if an implementation provides integer types with widths of 8, 16, 32, or 64 bits, it must define the corresponding **typedef** names.

14.8.2 Minimum-Width Integer Types

The **typedef** names **int_leastN_t** and **uint_leastN_t**, respectively, designate signed and unsigned integer types with a width of at least *N* bits, such that no signed integer type with lesser size has at least the specified width.

For example, **uint_least16_t** denotes an unsigned integer type with a width of at least 16 bits.

The following types are mandatory:

```
int_least8_t    int_least32_t    uint_least8_t    uint_least32_t
int_least16_t   int_least64_t    uint_least16_t   uint_least64_t
```

14.8.3 Fastest Minimum-Width Integer Types

The **typedef** names **int_fastN_t** and **uint_fastN_t**, respectively, designate the (usually) fastest signed and unsigned integer types with a width of at least *N* bits.

The following types are mandatory:

```
int_fast8_t    int_fast32_t    uint_fast8_t    uint_fast32_t
int_fast16_t   int_fast64_t    uint_fast16_t   uint_fast64_t
```

14.8.4 Integer Types Capable of Holding Object Pointers

The optional **intptr_t** and **uintptr_t** types, respectively, designate a signed and unsigned integer type with the property that any valid pointer to **void** can be converted to this type, then converted back to a pointer to **void** and the result will compare equal to the original pointer.

14.8.5 Greatest-Width Integer Types

The **intmax_t** and **uintmax_t** types, respectively, designate a signed integer type capable of representing any value of any signed integer type, and an unsigned integer type capable of representing any value of any unsigned integer type.

For each type declared in **<stdint.h>** conversion macros, which expand to the correct format specifiers, are defined for use with the formatted input/output functions such as *fprintf()* and *fscanf()*.

The *fprintf()* macros for signed integers are:

```
PRIdFASTN    PRIdMAX    PRIdPRT    PRIiLEASTN    PRIiN
PRIdLEASTN   PRIdN      PRIiFASTN   PRIiMAX      PRIiPRT
```

The PRId macros each expand to a string literal suitable for use as a *d* print conversion specifier, plus any needed qualifiers, to convert values of the types **int8_t**, **int16_t**, **int32_t**, or **int64_t**, respectively.

The PRIdLEAST macros each expand to a string literal suitable for use as a *d* print conversion specifier, plus any needed qualifiers, to convert values of the types **int_least8_t**, **int_least16_t**, **int_least32_t**, or **int_least64_t**, respectively.

The PRIdFAST macros each expand to a string literal suitable for use as a *d* print conversion specifier, plus any needed qualifiers, to convert values of the types **int_fast8_t**, **int_fast16_t**, **int_fast32_t**, or **int_fast64_t**, respectively.

The `PRIdMAX` macro expands to a string literal suitable for use as a `d` print conversion specifier, plus any needed qualifiers, to convert values of the type `intmax_t`.

The `PRIdPTR` macro expands to a string literal suitable for use as a `d` print conversion specifier, plus any needed qualifiers, to convert values of the type `intptr_t`.

The following example shows part of one possible implementation of these macros in an LP64 programming model:

```
#define PRId8      "hhd"
#define PRId16     "hd"
#define PRId32     "d"
#define PRId64     "ld"

#define PRIdFAST8  "hhd"
#define PRIdFAST16 "hd"
#define PRIdFAST32 "d"
#define PRIdFAST64 "ld"

#define PRIdLEAST8  "hhd"
#define PRIdLEAST16 "hd"
#define PRIdLEAST32 "d"
#define PRIdLEAST64 "ld"
```

Corresponding `fprintf()` macros are defined for unsigned integers (`PRIo`, `PRlu`, `PRIx`, and `PRIX`).

For `fscanf()`, the macro names start with `SCN` instead of `PRN`. (`SCNd` and `SCNi` for signed integers; `SCNo`, `SCNu`, `SCNx` for unsigned integers.)

A new structure type `imaxdiv_t` is defined. It is the type of structure returned by `imaxdiv()`.

The following function computes the absolute value of an integer `j`:

```
intmax_t imaxabs(intmax_t j);
```

The following function computes both the quotient and remainder in a single operation:

```
imaxdiv_t imaxdiv(intmax_t numer, intmax_t denom);
```

The following functions are equivalent to the `strtol` family of functions, except that the initial portion of the string is converted to `intmax_t` and `uintmax_t` representation, respectively:

```
intmax_t strtointmax(const char *restrict nptr,
                    char **restrict endptr, int base);
uintmax_t strtouintmax(const char *restrict nptr,
                      char **restrict endptr, int base);
```

The following functions are equivalent to the `wcstol` family of functions, except that the initial portion of the wide string is converted to `intmax_t` and `uintmax_t` representation, respectively:

```
intmax_t wcstointmax(const wchar_t *restrict nptr,
                    wchar_t **restrict endptr, int base);
uintmax_t wcstouintmax(const wchar_t *restrict nptr,
                      wchar_t **restrict endptr, int base);
```

14.8.6 Limits of Specified-Width Integer Types

The standard specifies the minimum and maximum limits for all of the types declared in the `<stdint.h>` header.

For example, the minimum value of an exact-width unsigned integer type is `{UINTn_MAX}`, where n is an unsigned decimal integer with no leading zeros, whose value is exactly 2^{n-1} .

14.8.7 Macros

The macros `INTN_C()` and `UINTN_C()` expand to a signed integer constant whose type and value is `int_leastN_t`, and an unsigned integer constant whose type and value is `uint_leastN_t`, respectively.

The macro `INTMAX_C()` expands to a integer constant whose type is `intmax_t`, and `UINTMAX_C()` expands to an unsigned integer constant whose type `uintmax_t`.

14.9 Complex Numbers

Support for complex numbers and complex number arithmetic is new, and was added as part of the effort to make the C language more attractive for general numerical programming. The underlying implementation of the complex types is explicitly stated to be Cartesian, rather than polar, for consistency with other programming languages. Thus values are interpreted as radians, not degrees.

The header `<complex.h>` contains the macro definitions and function declarations that support complex arithmetic.

Two new type specifiers were defined:

`_Complex`

`_Imaginary` (Only if an implementation supports a pure imaginary type.)

A new type qualifier `complex` (actually a macro which expands to `_Complex`) is used to denote a number as being a complex number.

Three complex types were defined:

- **float complex**
- **double complex**
- **long double complex**

The corresponding real type is the type obtained by deleting the type qualifier `complex` from the complex type name.

A complex type has the same representation and alignment requirements as an array type containing exactly two elements of the corresponding real type; the first element is equal to the real part, and the second element to the imaginary part, of the complex number.

There is no special syntax for constants; instead there is a new macro `_Complex_I`, which has a complex value whose real part is zero and whose imaginary part is x . Note that `_Complex_I*_Complex_I` has a value of -1 , but the type of that value is `complex`.

The standard reserves the keyword `_Imaginary` for use as a type-specifier in conjunction with the pure imaginary type. The macros `imaginary` and `Imaginary_I` are defined only if an

implementation supports a pure **imaginary** type. Such support is optional. See Annex G of the standard for further details. If defined, they expand to `_Imaginary` and a constant expression of type `const float _Imaginary` with the value of the imaginary unit.

The macro `I` expands to `_Imaginary_I`, if defined, else to `_Complex_I`. Thus a complex number constant (`3.0+4.0i`) could be written as either `3.0+4.0*I` or `3.0+4.0*_Complex_I`.

The choice of `'I'` instead of `'i'` for the imaginary unit was because of the widespread use of the identifier `'i'` for other purposes. A program can use a different identifier (for example, `'z'`) for the imaginary unit by undefining `'I'` and defining `'z'` as follows:

```
#include <complex.h>
#undef I
#define z _Imaginary_z
```

Annex G, which is marked informative, specifies complex arithmetic intended to be compatible with the IEC 60559:1989 standard real floating-point arithmetic. This annex was designated as informative because of insufficient prior art for normative status. An implementation claiming such conformance should define `__STDC_IEC_559_COMPLEX` to be 1.

The **pragma STDC CX_LIMITED_RANGE** can be used to indicate (ON) that the usual mathematical formulas for complex arithmetic may be used. Such formulas are problematic because of overflow, underflow, and handling of infinities.

The following new functions relate to complex arithmetic.

14.9.1 Trigonometric Functions

The complex arc cosine functions compute the complex arc cosine of z , with branch cuts outside the interval $[-1,+1]$ along the real axis.

```
double complex cacos(double complex z);
float complex cacosf(float complex z);
long double complex cacosl(long double complex z);
```

The complex arc sine functions compute the complex arc sine of z , with branch cuts outside the interval $[-1,+1]$ along the real axis.

```
double complex casin(double complex z);
float complex casinf(float complex z);
long double complex casinl(long double complex z);
```

The complex arc tangent functions compute the complex arc tangent of z , with branch cuts outside the interval $[-i,+i]$ along the imaginary axis.

```
double complex catan(double complex z);
float complex catanf(float complex z);
long double complex catanl(long double complex z);
```

The complex cosine functions compute the complex cosine of z .

```
double complex ccos(double complex z);
float complex ccosf(float complex z);
long double complex ccosl(long double complex z);
```

The complex sine functions compute the complex sine of z .

```
double complex csin(double complex z);
float complex csinf(float complex z);
long double complex csinl(long double complex z);
```

The complex tangent functions compute the complex tangent of z .

```
double complex ctan(double complex z);
float complex ctanf(float complex z);
long double complex ctanl(long double complex z);
```

14.9.2 Hyperbolic Functions

The complex arc hyperbolic cosine functions compute the complex arc hyperbolic cosine of z , with a branch cut at values less than 1 along the real axis.

```
double complex cacosh(double complex z);
float complex cacoshf(float complex z);
long double complex cacoshl(long double complex z);
```

The complex arc hyperbolic sine functions compute the complex arc hyperbolic sine of z , with branch cuts outside the interval $[-i,+i]$ along the imaginary axis.

```
double complex casinh(double complex z);
float complex casinhf(float complex z);
long double complex casinhl(long double complex z);
```

The complex arc hyperbolic tangent functions compute the complex arc hyperbolic tangent of z , with branch cuts outside the interval $[-1,+1]$ along the real axis.

```
double complex catanh(double complex z);
float complex catanhf(float complex z);
long double complex catanhl(long double complex z);
```

The complex hyperbolic cosine functions compute the complex hyperbolic cosine of z .

```
double complex ccosh(double complex z);
float complex ccoshf(float complex z);
long double complex ccoshl(long double complex z);
```

The complex hyperbolic sine functions compute the complex hyperbolic sine of z .

```
double complex csinh(double complex z);
float complex csinhf(float complex z);
long double complex csinhl(long double complex z);
```

The complex hyperbolic tangent functions compute the complex hyperbolic tangent of z .

```
double complex ctanh(double complex z);
float complex ctanhf(float complex z);
long double complex ctanhl(long double complex z);
```

14.9.3 Exponential and Logarithmic Functions

The complex exponential functions compute the complex base- e exponential of z .

```
double complex cexp(double complex z);
float complex cexpf(float complex z);
long double complex cexpl(long double complex z);
```

The complex natural logarithm functions compute the complex natural logarithm of z , with a branch cut along the negative real axis.

```
double complex clog(double complex z);
float complex clogf(float complex z);
long double complex clogl(long double complex z);
```

14.9.4 Power and Absolute-Value Functions

The complex absolute value functions compute the modulus of x .

```
double complex cabs(double complex x);
float complex cabsf(float complex x);
long double complex cabsl(long double complex x);
```

The complex power functions compute the complex power function $x^{**}y$, with a branch cut for the first parameter along the negative real axis.

```
double complex cpow(double complex x, double complex y);
float complex cpowf(float complex x, float complex y);
long double complex cpowl(long double complex x,
    long double complex y);
```

The complex square root functions compute the complex square root of z , with a branch cut along the negative real axis.

```
double complex csqrt(double complex z);
float complex csqrtf(float complex z);
long double complex csqrtl(long double complex z);
```

14.9.5 Manipulation Functions

The complex argument functions compute the argument of z , with a branch cut along the negative real axis.

```
double complex carg(double complex z);
float complex cargf(float complex z);
long double complex cargl(long double complex z);
```

The complex imaginary functions compute the imaginary part of z .

```
double complex cimag(double complex z);
float complex cimagf(float complex z);
long double complex cimagl(long double complex z);
```

The complex conjugate functions compute the complex conjugate of z , by reversing the sign of its imaginary part.

```
double complex conj(double complex z);
float complex conjf(float complex z);
long double complex conjl(long double complex z);
```

The complex projection functions compute a projection of z onto the Riemann sphere.

```
double complex cproj(double complex z);
float complex cprojf(float complex z);
long double complex cprojl(long double complex z);
```

The complex real functions compute the real part of z .

```
double creal(double complex z);
float crealf(float complex z);
long double creall(long double complex z);
```

Note that no errors are defined for any of the above functions.

14.10 Other Mathematical Changes

The standard extended the mathematical support via `<math.h>` by providing versions of functions to support **float** and **long double** as well as the existing double floating type functions.

The functions `ecvt()`, `fcvt()`, and `gcvt()` were dropped from the standard since their capability is available using the `sprintf()` function.

The **pragma STDC FP_CONTACT** indicates to an implementation whether it is allowed (ON) or disallowed (OFF) to contract expressions; that is, evaluated as though an expression is an atomic operation, thereby omitting certain rounding errors.

The macro `NAN` is defined only if an implementation supports quiet NaNs.

14.10.1 Classification Macros

The following are defined for use with classification macros:

<code>FP_NAN</code>	The floating-point number <code>x</code> is “Not a Number”.
<code>FP_INFINITE</code>	The value of the number is either plus or minus infinity.
<code>FP_ZERO</code>	The value of the number is either plus or minus zero.
<code>FP_SUBNORMAL</code>	The number is in denormalized format.
<code>FP_NORMAL</code>	There is nothing special about the number.

The macro `fpclassify()` classifies its argument as either NaN, infinite, normal, subnormal, zero, or into another implementation-defined category.

```
int fpclassify(real-floating x);
```

The macro `isfinite()` determines whether its argument has a finite value (zero, subnormal, or normal, and not infinite or NaN).

```
int isfinite(real-floating x);
```

The macro `isinf()` determines whether its argument value is an infinity (positive or negative).

```
int isinf(real-floating x);
```

The macro `isnan()` determines whether its argument value is a NaN.

```
int isnan(real-floating x);
```

The macro `isnormal()` determines whether its argument value is normal (neither zero, subnormal, infinite, nor NaN).

```
int isnormal(real-floating x);
```

The macro `signbit()` determines whether the sign of its argument value is negative.

```
int signbit(real-floating x);
```

14.10.2 Trigonometric Functions

The following functions compute the arc cosine, arc sin, and arctan of x , respectively:

```
float acosf(float x);
long double acosl(long double x);
float asinf(float x);
long double asinl(long double x);
long double tanl(long double x);
```

The following functions compute the arc tangent of y/x :

```
float atan2f(float y, float x);
long double atan2l(long double y, long double x);
```

The following functions compute the cosine, sin, and tangent of x , respectively:

```
float cosf(float x);
long double cosl(long double x);
float sinf(float x);
long double sinl(long double x);
float tanf(float x);
long double tanl(long double x);
```

14.10.3 Hyperbolic Functions

The following functions compute the arc hyperbolic cosine of x :

```
float acoshf(float x);
long double acoshl(long double x);
```

The following functions compute the arc hyperbolic sine of x :

```
float asinhf(float x);
long double asinhl(long double x);
```

The following functions compute the arc hyperbolic tangent of x :

```
float atanhf(float x);
long double atanhhl(long double x);
```

The following functions compute the hyperbolic cosine of x :

```
float coshf(float x);
long double coshl(long double x);
```

The following functions compute the hyperbolic sine of x :

```
float sinhlf(float x);
long double sinhl(long double x);
```

The following functions compute the hyperbolic tangent of x :

```
float tanhf(float x);
long double tanhl(long double x);
```

14.10.4 Exponential and Logarithmic Functions

The following functions compute the base-e exponential of x :

```
float expf(float x);
long double expl(long double x);
```

The following functions compute the base-2 exponential of x :

```
double exp2(double x);
float exp2f(float x);
long double exp2l(long double x);
```

The following functions compute the base-e exponential of $(x-1)$:

```
float expm1f(float x);
long double expm1l(long double x);
```

The following functions break a floating-point number into a normalized fraction and an integral power of 2:

```
float frexpf(float value, int *exp);
long double frexpl(long double value, int *exp);
```

The following functions extract the exponent of x :

```
int ilogbf(float x);
int ilogbl(long double x);
```

The following functions multiply a floating-point number by an integral power of 2:

```
float ldexpf(float x, int exp);
long double ldexpl(long double x, int exp);
```

The following functions compute the natural logarithm of x :

```
float logf(float x);
long double logl(long double x);
```

The following functions compute the base-10 logarithm of x :

```
float log10f(float x);
long double log10l(long double x);
```

The following functions compute the natural logarithm of $(x+1)$:

```
float log1pf(float x);
long double log1pl(long double x);
```

The following functions compute the base-2 logarithm of x :

```
double log2(double x);
float log2f(float x);
long double log2l(long double x);
```

The following functions extract the exponent of x :

```
float logbf(float x);
long double logbl(long double x);
```

The following functions break x into integral and fractional parts:

```
float modff(float x, float *iptr);
long double modfl(long double x, long double *iptr);
```

The following functions compute $x^{\text{FLT_RADIX}^n}$ efficiently:

```
double scalbn(double x, int n);
float scalbnf(float x, int n);
long double scalbnl(long double x, int n);
double scalbln(double x, long int n);
float scalblnf(float x, long int n);
long double scalblnl(long double x, long int n);
```

The following functions compute the real cube root of x :

```
double cbrt(double x);
float cbrtf(float x);
long double cbrtl(long double x);
```

The following functions compute the absolute value of x :

```
float fabsf(float x);
long double fabsl(long double x);
```

The following functions compute the square root of the sum of the squares of x and y :

```
float hypotf(float x, float y);
long double hypotl(long double x, long double y);
```

The following functions compute x raised to the power of y :

```
float powf(float x, float y);
long double powl(long double x, long double y);
```

The following functions compute the non-negative square root of x :

```
float sqrtf(float x);
long double sqrtl(long double x);
```

The following functions compute the error function of x :

```
float erff(float x);
long double erfl(long double x);
```

The following functions compute the natural logarithm of the absolute value of the gamma function of x :

```
float lgammaf(float x);
long double lgammal(long double x);
```

The following functions compute the (true) gamma function of x :

```
double tgamma(double x);
float tgammaf(float x);
long double tgammal(long double x);
```

14.10.5 Nearest Integer Functions

The following functions compute the smallest integer value not less than x :

```
float ceilf(float x);
long double ceill(long double x);
```

The following functions compute the largest integer value not greater than x :

```
float floorf(float x);
long double floorl(long double x);
```

The following functions round x to an integer value in floating-point format, using the current rounding direction and without raising the inexact floating-point exception:

```
double nearbyint(double x);
float nearbyintf(float x);
long double nearbyintl(long double x);
```

The following functions round x to an integer value in floating-point format, using the current rounding direction and may raise the inexact floating-point exception if the result differs in value from the argument:

```
float rintf(float x);
long double rintl(long double x);
```

The following functions round x to the nearest integer value, rounding according to the current rounding direction:

```
long int lrint(double x);
long int lrintf(float x);
long int lrintl(long double x);
long long int llrint(double x);
long long int llrintf(float x);
long long int llrintl(long double x);
```

The following functions round x to the nearest integer value in floating-point format, rounding halfway cases away from zero, regardless of the current rounding direction:

```
double round(double x);
float roundf(float x);
long double roundl(long double x);
```

The following functions round x to the nearest integer value, rounding halfway cases away from zero, regardless of the current rounding direction:

```
long int lround(double x);
long int lroundf(float x);
long int lroundl(long double x);
long long int llround(double x);
long long int llroundf(float x);
long long int llroundl(long double x);
```

The following functions round x to the integer value, in floating format, nearest to but no larger in magnitude than x :

```
double trunc(double x);
float truncf(float x);
long double trunc1(long double x);
```


14.10.6 Remainder Functions

The following functions compute the floating-point remainder of x/y :

```
float fmodf(float x, float y);
long double fmodl(long double x, long double y);
```

The following functions compute the IEC 60559: 1989 standard remainder $x \text{ REM } y$:

```
float remainderf(float x, float y);
long double remainderl(long double x, long double y);
```

The following functions shall compute the same remainder as the remainder family of functions, but in a different manner:

```
double remquo(double x, double y, int *quo);
float remquof(float x, float y, int *quo);
long double remquol(long double x, long double y, int *quo);
```

14.10.7 Manipulation Functions

The following functions produce a value with the magnitude of *and the sign of* y :

```
double copysign(double x, double y);
float copysignf(float x, float y);
long double copysignl(long double x, long double y);
```

The following functions return a quiet NaN, if available, with content indicated by *tagp*:

```
double nan(const char *tagp);
float nanf(const char *tagp);
long double nanl(const char *tagp);
```

The following functions determine the next representable value:

```
float nextafterf(float x, float y);
long double nextafterl(long double x, long double y);
```

The following functions are equivalent to the *nextafter* functions, except that the second parameter has type **long double** and the functions return y converted to the type of the function if x equals y :

```
double nexttoward(double x, long double y);
float nexttowardf(float x, long double y);
long double nexttowardl(long double x, long double y);
```

The following functions determine the positive difference between their arguments:

```
double fdim(double x, double y);
float fdimf(float x, float y);
long double fdiml(long double x, long double y);
```

The following functions determine the maximum numeric value of their arguments:

```
double fmax(double x, double y);
float fmaxf(float x, float y);
long double fmaxl(long double x, long double y);
```

If the optional macros `FP_FAST_FMA`, `FP_FAST_FMAF`, and `FP_FAST_FMAL` are defined, it indicates that the corresponding *fma()* function executes at least as fast as a multiply and an

add of double operands.

The following functions determine the minimum numeric value of their arguments:

```
double fmin(double x, double y);
float fminf(float x, float y);
long double fminl(long double x, long double y);
```

The following functions compute $(x*y)+z$, rounded as one ternary operation:

```
double fma(double x, double y, double z);
float fmaf(float x, float y, float z);
long double fmal(long double x, long double y,
                 long double z);
```

14.10.8 Comparison Macros

The *isgreater()* macro tests whether *x* is greater than *y*.

```
int isgreater(real-floating x, real-floating y);
```

The *isgreaterequal()* macro tests whether *x* is greater than or equal to *y*.

```
int isgreaterequal(real-floating x, real-floating y);
```

The *isless()* macro tests whether *x* is less than *y*.

```
int isless(real-floating x, real-floating y);
```

The *islessequal()* macro tests whether *x* is less than or equal to *y*.

```
int islessequal(real-floating x, real-floating y);
```

The *islessgreater()* macro tests whether *x* is less than or greater than *y*.

```
int islessgreater(real-floating x, real-floating y);
```

The *isunordered()* macro tests whether *x* and *y* are unordered.

```
int isunordered(real-floating x, real-floating y);
```

Note: Annex F (normative) was added to specify the IEC 60559:1989 standard floating-point arithmetic. An implementation that defines `__STDC_IEC_559__` must conform to the specification detailed in this annex.

14.11 Floating-Point Environment Support

The header `<fenv.h>` declares the types, and defines the macros and functions that support access to an implementation's floating-point environment.

Two types are declared:

fenv_t Represents the entire floating-point environment.

fexcept_t Represents the collective floating-point status flags.

The **pragma** directive has three reserved forms, all starting with the preprocessor token **STDC**. These are used to specify certain characteristics of the floating-point support to comply with the IEC 60559: 1989 standard.

The **pragma STDC FENV_ACCESS** provides the means of informing an implementation when a program might access the floating-point environment.

For example:

```
double a;
#pragma STDC FENV_ACCESS ON
a = 1.0 + 2.0;
#pragma STDC FENV_ACCESS OFF
```

14.11.1 Exceptions

The following function clears the supported floating-point exceptions:

```
void feclearexcept(int excepts);
```

The following function stores an implementation-dependent representation of the states of the floating-point status flags:

```
void fegetexceptflag(fexcept_t *flagp, int excepts);
```

The following function raises the supported floating-point exceptions represented by its argument:

```
void feraiseexcept(int excepts);
```

The following function sets the floating-point status flags:

```
void fesetexceptflag(const fexcept_t *flagp, int excepts);
```

The following function tests which of a specified subset of the floating-point exception flags are currently set:

```
int fetestexcept(int excepts);
```

Each of the following floating-point exception macros is defined if an implementation supports these functions:

```
FE_DIVBYZERO
FE_INEXACT
FE_INVALID
FE_OVERFLOW
FE_UNDERFLOW
FE_ALL_EXCEPT (Bitwise OR of all the other macros.)
```

Additional implementation-defined floating-point exceptions, with macro definitions beginning

with `FE_` and an uppercase letter, may also be defined by an implementation.

14.11.2 Rounding

The following functions respectively set and return the current rounding direction:

```
int fesetround(int round);
int fegetround(void);
```

Each of the following floating-point macros is defined if an implementation supports these functions:

```
FE_DOWNWARD
FE_TONEAREST
FE_TOWARDZERO
FE_UPWARD
```

Additional implementation-defined rounding directions, with macro definitions beginning with `FE_` and an uppercase letter, may also be defined by an implementation.

14.11.3 Environment

The following functions respectively set and return the floating-point environment function:

```
void fesetenv(const fenv_t *envp);
void fegetenv(fenv_t *envp);
```

The following function saves the currently raised floating-point exception(s), installs the floating-point environment represented by the object pointed to by `envp`, and then raises the saved floating-point exception(s):

```
void feupdateenv(const fenv_t *envp);
```

The following function saves the current floating-point environment in the object pointed to by `envp`, clears the floating-point status flags, and then installs a non-stop (continue on floating-point exceptions) mode, if available, for all floating-point exceptions:

```
int feholdexcept(fenv_t *envp);
```

The macro `FE_DFL_ENV` represents the default floating-point environment; that is, the one installed at program startup. It can be used as an argument with the above functions and is of type `*const fenv_t`.

14.12 Type-Generic Math

Type-generic macros may enable the writing of more portable code, and reduce need for casting and suffixing when porting applications to new platforms.

The header `<tgmath.h>` includes the headers `<math.h>` and `<complex.h>` and defines numerous type-generic macros. Except for `modf`, there is a type-generic macro for each of the functions in `<math.h>` and `<complex.h>` that do not have an `'f'` (**float**) or `'l'` (**long double**) suffix and have one or more parameters whose corresponding real type is **double**.

Such parameters are called generic parameters.

Use of a type-generic macro invokes a function whose corresponding real type and type domain are determined by the arguments for the generic parameters. The real type is determined as follows:

1. First, if any argument for generic parameters is a **long double**, the real type is **long double**.
2. Otherwise, if any argument for generic parameters is a double or an integer type, the real type is **double**.
3. Otherwise, the real type is **float**.

Type-generic macros that accept complex arguments also accept imaginary arguments. If an argument is imaginary, the macro expands to an expression whose type is **real**, **imaginary**, or **complex**, as appropriate for the particular function.

14.12.1 Unsuffixed Functions With a C-Prefixed Counterpart

For each unsuffixed function in `<math.h>` for which there is a function in `<complex.h>` with the same name except for a `'c'` prefix, the corresponding type-generic macro for both functions has the same name as the function in `<math.h>`.

For example, the type-generic macro for `tan()` and `ctan()` is `tan`.

If at least one argument for a generic parameter is complex, then use of the macro invokes a complex function; otherwise, a real function is invoked.

14.12.2 Unsuffixed Functions Without a C-Prefixed Counterpart

For each unsuffixed function in `<math.h>` for which there is not a function in `<complex.h>` with the same name but having a `'c'` prefix, the corresponding type-generic macro for both functions has the same name as the function in `<math.h>`. If all arguments for generic parameters are real, then use of the macro invokes a real function; otherwise, use of the macro results in undefined behavior. Examples of such functions include `fdim()` and `lround()`.

For each unsuffixed function in `<complex.h>` for which there is not a function in `<math.h>` with the same name but without a `'c'` prefix, the corresponding type-generic macro for both functions has the same name as the function in `<complex.h>`. Use of the macro with any real or complex argument invokes a complex function.

14.13 Other Library Changes

A number of new functions were added to the standard, prototypes for many functions now contain the new keyword **restrict** as part of some parameter declarations, and a number of functions had their definition clarified or extended.

atoll()

A numeric conversion function for the conversion of a string to a **long long int** representation.

```
long long int atoll(const char *nptr);
```

_Exit()

This function causes normal program termination to occur and control to be returned to the host environment without triggering signals or *atexit()* registered functions.

This function name (rather than *_exit()*) was chosen to avoid potential conflict with existing practice.

fpos_t

The description of **fpos_t** was changed to exclude array type objects.

isblank()

This function tests whether *c* is a character of class **blank** in a program's current locale.

```
int isblank(int c);
```

iswblank()

This function tests whether *wc* is a wide-character which is a member of the class **blank** in the program's current locale.

```
int iswblank(wint_t wc);
```

llabs()

In a similar manner to its counterparts *abs()* and *labs()*, this function computes the absolute value of an integer.

```
long long int llabs(long long int j);
```

lldiv()

In a similar manner to its counterparts *div()* and *ldiv()*, this function returns a structure of type **lldiv_t** which contains both the quotient and the remainder, each of which is of type **long long int**.

```
lldiv_t lldiv(long long int numer, long long int denom);
```

localeconv()

The standard added the following members to the **lconv** structure (defined in **<locale.h>**) to assign with long-standing POSIX practice and to permit additional flexibility with internationally formatted monetary quantities:

char p_cs_precedes	Set to 1 or 0 if the currency_symbol respectively precedes or succeeds the value for a non-negative locally formatted monetary quantity.
char n_cs_precedes	Set to 1 or 0 if the currency_symbol respectively precedes or succeeds the value for a negative locally formatted monetary quantity.
char p_sep_by_space	Set to a value indicating the separation of the currency_symbol , the sign string, and the value for a non-negative locally formatted monetary quantity.
char n_sep_by_space	Set to a value indicating the separation of the currency_symbol , the sign string, and the value for a negative locally formatted monetary quantity.
char p_sign_posn	Set to a value indicating the positioning of the positive_sign for a non-negative locally formatted monetary quantity.
char n_sign_posn	Set to a value indicating the positioning of the negative_sign for a negative locally formatted monetary quantity.

printf(), fprintf(), sprintf()

New length modifiers were added to the standard:

- hh Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **signed char** or **unsigned char** argument; or that a following **n** conversion specifier applies to a pointer to a **signed char** argument. This modifier enables character types to be treated the same as all other integer types.
- ll Added to support the new **long long int** type. Specifies that a following **d**, **I**, **o**, **u**, **x**, or **X** conversion specifier applies to a **long long int** or **unsigned long long int** argument; or that a following **n** conversion specifier applies to a pointer to a **long long int** argument.

The maximum number of characters that can be produced by any single conversion was increased from 509 characters (C89) to 4 095 characters.

realloc()

The description of this function was changed to make it clear that the pointed-to object is deallocated, a new object is allocated, and the content of the new object is the same as that of the old object up to the lesser of the two sizes.

scanf(), fscanf(), sscanf()

The **hh** and **ll** length modifiers (see *printf()* above) were added.

Also the conversion modifiers **a** and **A** were added with **A** being equivalent to **a**.

These conversion modifiers match an optionally signed floating-point number, infinity, or NaN, whose format is the same as expected for the subject sequence of the *strtod()* function. The corresponding argument shall be a pointer to floating.

The behavior of the *sscanf()* function on encountering the end of a string has been clarified.

setvbuf()

The function prototype was changed to include the **restrict** type qualifier:

```
int setvbuf(FILE *restrict stream, char *restrict buf,
            int type, size_t size);
```

In previous revisions of the standard it was not clear about what, if anything, *size* means when *buf* is a null pointer. The standard now warns that *size* might not be ignored, so portable programs should supply a reasonable value.

snprintf()

This function was added to the standard to address the problem of *sprintf()* potentially overrunning an output buffer. It is equivalent in functionality to *sprintf()* except that it performs bounds checking on the output array. Extra characters are discarded and a null character is written at the end of the characters actually written to the array.

strftime()

The definition of this function was changed to incorporate additional conversion specifiers defined in the ISO POSIX-1:1996 standard, including %C, %D, %e, %F, %g, %G, %h, %n, %r, %R, %t, %T, %u, and %V, as well as the E and O modifiers.

strtod(), strtodf(), strtold ()

The following two functions were added to the standard:

```
float strtodf(const char *restrict nptr,
              char **restrict endptr);
long double strtold (const char *restrict nptr,
                     char **restrict endptr);
```

In a similar manner to their counterpart, the *strtod()* function, these functions convert the initial portion of the string pointed to by *nptr* to **float** and **long double** representation, respectively. Support for subject sequences relating to floating-point (NaN, INF, and so on) was also added.

strtoll(), strtoull()

The following two functions were added to the standard:

```
long long int strtoll(const char *restrict nptr,
                     char **restrict endptr, int base);
unsigned long int strtoull(const char *restrict nptr,
                          char **restrict endptr, int base);
```

In a similar manner to their counterparts, the *strtoll()* and *strtoull()* functions, these functions convert the initial portion of the string pointed to by *nptr* to **long long int** and **unsigned long int** representation, respectively.

tmpnam()

The previous standard had a serious flaw regarding this function. If the function were called fewer than {TMP_MAX} times but was unable to generate a suitable string because every potential string named an existing file, there was no way to report failure and no undefined behavior; hence there was no option other than to never return.

This standard resolved this issue by allowing the function to return a null pointer when it cannot generate a suitable string and by specifying that {TMP_MAX} is the number of potential strings, any or all of which may name existing files and thus not be suitable return values.

Note: This is a quiet change in the standard. Programs that call this function without checking for a null return value may produce undefined behavior.

ungetc()

The standard deprecated the use of this function on a binary stream at the beginning of the file.

vfscanf()

The following functions are functionally the same as *scanf()*, *fscanf()*, and *sscanf()* respectively, except that instead of being called with a variable number of arguments, they are called with an argument list:

```
int vscanf(const char *restrict format, va_list arg);
int vfscanf(FILE *restrict stream,
            const char *restrict format, va_list arg);
int vsscanf(const char *restrict s,
            const char *restrict format, va_list arg);
```

vfwscanf()

The following functions are functionally the same as *fwscanf()*, *swscanf()*, and *wscanf()* respectively, except that instead of being called with a variable number of arguments, they are called with an argument list:

```
int vfwscanf(FILE *restrict stream,
            const wchar_t *restrict format, va_list arg);
int vswscanf(const wchar_t *restrict s,
            const wchar_t *restrict format, va_list arg);
int vwscanf(const wchar_t *restrict format, va_list arg);
```

14.13.1 Wide-String Numeric Conversion Functions

The following functions were added to the existing wide-string numeric conversion functions:

```
float wcstof(const wchar_t *restrict nptr,
            wchar_t **restrict endptr);
long double wcstold(const wchar_t *restrict nptr,
            wchar_t **restrict endptr);
long long int wcstoll(const wchar_t *restrict nptr,
            wchar_t **restrict endptr, int base);
long long int wcstoull(const wchar_t *restrict nptr,
            wchar_t **restrict endptr, int base);
```

14.14 Annexes

A number of new normative and informative annexes were added to the standard and some exiting annexes were modified.

Annexes A, B, and E were modified to include the new keywords, universal character names, types, implementation limits, macros and functions, and other changes to the C language.

Annex F (normative) was added to specify the IEC 60559:1989 standard floating-point arithmetic. An implementation that defines `__STDC_IEC_559__` must conform to the specification detailed in this annex.

Annex G (informative) was added to specify recommended IEC 60559:1989 standard-compatible complex arithmetic. An implementation that defines `__STDC_IEC_559_COMPLEX__` should conform to the specification detailed in this annex. It is non-normative because there were few existing implementations at the time this standard was approved.

Annex H (informative) describes the extent of support in this standard for language-independent arithmetic as specified in the ISO/IEC 10967-1:1994 standard. This annex was added, however, because all programming languages covered by ISO/IEC JTC1 SC22 standards are expected to review the ISO/IEC 10967-1:1994 standard and incorporate and further define the binding between that standard and each programming language.

Index

#line	371
<aioh>	47
<arpa/inet.h>	47
<assert.h>	46
<cpio.h>	47
<ctype.h>	46
<dirent.h>	47
<dlfcn.h>	47
<errno.h>	46-47
<fcntl.h>	47
<float.h>	46
<fmtmsg.h>	47
<ftw.h>	47
<grp.h>	47
<iconv.h>	47
<langinfo.h>	47
<libgen.h>	47
<limits.h>	46-47
<locale.h>	46
<math.h>	46
<monetary.h>	47
<mqueue.h>	47
<ndbm.h>	47
<netdb.h>	47
<netinet/in.h>	47
<netinet/tcp.h>	47
<nl_types.h>	47
<poll.h>	47
<pthread.h>	47
<pwd.h>	47
<sched.h>	47
<search.h>	47
<semaphore.h>	47
<setjmp.h>	46-47
<signal.h>	46-47
<stdarg.h>	46
<stddef.h>	46
<stdio.h>	46-47
<stdlib.h>	46
<string.h>	46
<strings.h>	47

<stropts.h>.....	47
<sys/ipc.h>	47
<sys/mman.h>	47
<sys/msg.h>	47
<sys/resource.h>	47
<sys/select.h>	47
<sys/sem.h>	47
<sys/shm.h>	47
<sys/socket.h>.....	47
<sys/stat.h>	47
<sys/statvfs.h>.....	47
<sys/time.h>	47
<sys/timeb.h>	47
<sys/times.h>	47
<sys/types.h>	47
<sys/uio.h>	47
<sys/utsname.h>	47
<sys/wait.h>.....	47
<syslog.h>	47
<tar.h>	47
<termios.h>	47
<time.h>	46-47
<ulimit.h>.....	47
<unistd.h>.....	47
<utime.h>.....	47
<utmpx.h>	47
_exit.....	101
_Exit	390
_POSIX_ADVISORY_INFO.....	33
_POSIX_ASYNCHRONOUS_IO	22
_POSIX_BARRIERS.....	22
_POSIX_CLOCK_SELECTION.....	22
_POSIX_CPUTIME	33
_POSIX_C_SOURCE	10, 46
_POSIX_FSYNC	33
_POSIX_MAPPED_FILES	22
_POSIX_MEMLOCK	33
_POSIX_MEMLOCK_RANGE	33
_POSIX_MEMORY_PROTECTION	22
_POSIX_MESSAGE_PASSING	33
_POSIX_MONOTONIC_CLOCK	33
_POSIX_PRIORITIZED_IO.....	33
_POSIX_PRIORITY_SCHEDULING.....	33
_POSIX_READER_WRITER_LOCKS	22
_POSIX_REALTIME_SIGNALS	22
_POSIX_SEMAPHORES	22
_POSIX_SHARED_MEMORY_OBJECTS.....	33
_POSIX_SOURCE	46
_POSIX_SPAWN.....	33
_POSIX_SPIN_LOCKS.....	22
_POSIX_SPORADIC_SERVER	33
_POSIX_SYNCHRONIZED_IO	33

_POSIX_THREADS	22
_POSIX_THREAD_CPUTIME	33
_POSIX_THREAD_PRIORITY_SCHEDULING	33
_POSIX_THREAD_PRIO_INHERIT	33
_POSIX_THREAD_PRIO_PROTECT	33
_POSIX_THREAD_SAFE_FUNCTIONS	22
_POSIX_THREAD_SPORADIC_SERVER	33
_POSIX_TIMEOUTS	22
_POSIX_TIMERS	22
_POSIX_TRACE	34
_POSIX_TRACE_EVENT_FILTER	34
_POSIX_TRACE_INHERIT	34
_POSIX_TRACE_LOG	34
_POSIX_TYPED_MEMORY_OBJECTS	33
_setjmp	101
_XOPEN_SOURCE	10, 14, 47
_XOPEN_UNIX	10
absolute-value function	379
acosf	104
acoshf	104
acoshl	104
acosl	104
adb, rationale for omission	40
addch	52
addchnstr	52
addchstr	52
addnstr	52
addnwstr	52
address information	146
addstr	52
addwstr	52
add_wch	52
add_wchnstr	52
add_wchstr	52
admin	37-39
advanced realtime	33
ADVANCED REALTIME	117, 204-209, 215, 352
advanced realtime threads	33
ADVANCED REALTIME THREADS	226
alias	38
anonymous aggregate	366
ar	38
argument macros	371
arithmetic	370
array declaration	368
array type compatibility	368
as, rationale for omission	40
asa	38
asctime_r	107
asinf	107
asinhf	107
asinh	107

asinl	107
async-signal-safe	10
at	38
atan2f	108
atan2l	108
atanf	108
atanhf	108
atanhl	108
atanl	108
atoll	109, 390
attroff	52
attron	52
attrset	52
attr_get	52
attr_off	52
attr_on	52
attr_set	52
awk	9, 14, 38
banner, rationale for omission	40
basename	38
batch	38
baudrate	52
bc	38
beep	52
bg	38
bkgd	52
bkgdset	52
bkgrnd	52
bkgrndset	52
block	369
Boolean	365
border	52
border_set	52
box	52
box_set	52
C-language development tools	38
C-language support	9
c99	9, 38
cabsf	110
cabsl	110
cacosf	111
cacoshf	111
cacoshl	111
cacosl	111
cal	38
calendar, rationale for omission	40
cancel, rationale for omission	40
can_change_color	52
cargf	111
cargl	111
case-sensitive identifiers	371
casinf	111

casinhf	112
casinhl	112
casinl	111
cat.....	38
catanf.....	112
catanhf.....	112
catanhl.....	112
catanl.....	112
catclose	15
catgets.....	15
catopen.....	15
cbreak.....	52
cbrtf	113
cbrtl.....	113
ccosf	113
ccoshf	114
ccoshl	114
ccosl	113
cd.....	38
ceilf	114
ceill	114
cexpf	114
cexpl	114
cflow	38-39
change history	21, 39, 48, 53
chgat.....	52
chgrp	38
chmod.....	38
chown	38
chroot, rationale for omission	41
cimagf.....	117
cimagl.....	117
cksum	38
classification macro	380
clear.....	52
clearok.....	52
clock_gettime.....	118
clock_settime.....	118
clogf.....	118
clogl.....	118
clrtoobot.....	52
clrtoeol.....	52
cmp.....	38
col, rationale for omission.....	41
COLORS	52
color_content.....	52
COLOR_PAIR.....	52
COLOR_PAIRS	52
color_set.....	52
COLS.....	52
comm.....	38
command.....	38

comment.....	370
common devices.....	7
common directories.....	7
comparison macro.....	386
complex number.....	376
compound literal.....	366
compress.....	38
compression utilities.....	38
conjf.....	120
conjl.....	120
copysignf.....	120
copysignl.....	120
copywin.....	52
cosf.....	120
coshf.....	121
coshl.....	121
cosl.....	120
cp.....	38
cpio, rationale for omission.....	41
cpowf.....	121
cpowl.....	121
cpp, rationale for omission.....	41
cprojf.....	121
cprojl.....	121
crealf.....	121
creall.....	121
crontab.....	38
CRYPT.....	122, 128, 254
csinf.....	122
csinhf.....	122
csinhl.....	122
csinl.....	122
csplit.....	9, 38
csqrtf.....	123
csqrtl.....	123
ctags.....	38-39
ctanf.....	123
ctanhf.....	123
ctanhl.....	123
ctanl.....	123
ctime_r.....	124
cu, rationale for omission.....	41
curscr.....	52
curs_set.....	52
cur_term.....	52
cut.....	38
cxref.....	38-39
date.....	38
daylight.....	282
DBM.....	124
dbm_close.....	124
dbm_delete.....	124

dbm_error.....	124
dbm_fetch.....	124
dbm_firstkey.....	124
dbm_nextkey.....	124
dbm_open.....	124
dbm_store.....	124
dc, rationale for omission.....	41
dd.....	38
decimal integer constant.....	367
def_prog_mode.....	52
def_shell_mode.....	52
delay_output.....	52
delch.....	52
deleteln.....	52
delscreen.....	52
delta.....	37-39
delwin.....	52
del_curterm.....	52
derwin.....	52
designated initializer.....	366
DEVELOPMENT.....	37-38, 81
df.....	38
diff.....	38
DIR.....	119
dircmp, rationale for omission.....	41
dirname.....	38
dis, rationale for omission.....	41
doudate.....	52
dprintf.....	145
du.....	38
dup2.....	127
dupwin.....	52
echo.....	38, 52
echochar.....	52
echo_wchar.....	52
ed.....	9, 38
emacs, rationale for omission.....	41
empty argument macros.....	371
encryption.....	32
endwin.....	52
ENHANCED CURSES.....	14
enumeration specifier.....	368
env.....	38
environment.....	388
environment variables.....	7
erand48.....	126
erase.....	52
erasechar.....	52
erasewchar.....	52
erfcf.....	130
erfcl.....	130
erff.....	130

erfl	130
errno	369
error descriptions.....	154
error numbers.....	10
ex.....	9, 38
exact-width integer type.....	373
exception	387
execl	131
execle	131
execlp	131
execv	131
execve	131
execvp	131
exp2f.....	132
exp2l.....	132
expand.....	38
expf.....	132
expl.....	132
expm1f.....	132
expm1l.....	132
exponential function.....	378, 382
expr.....	9, 38
external variable	
quick reference	55
fabsf.....	133
fabsl.....	133
faccessat	103
false.....	38
fastest minimum-width integer type	374
fc.....	38
fchmodat.....	115
fchownat	116
fdimf.....	135
fdiml.....	135
feature group	32
feature test macros.....	10, 46
fesetenv	137
fesetexceptflag	137
fesetround.....	137
fexecve	131
FE_ALL_EXCEPT	387
FE_DIVBYZERO	387
FE_DOWNWARD.....	388
FE_INEXACT	387
FE_INVALID.....	387
FE_OVERFLOW.....	387
FE_TONEAREST	388
FE_TOWARDZERO	388
FE_UNDERFLOW.....	387
FE_UPWARD.....	388
fg.....	38
FIFO	192

file.....	38
FILE.....	117, 134
file access.....	9
file format notation.....	13
filter.....	52
find.....	38
flash.....	52
flexible array member.....	368
floating-point constant.....	370
floating-point environment.....	387
floor.....	141
floorl.....	141
flushinp.....	52
fmaf.....	141
fmal.....	141
fmaxf.....	142
fmaxl.....	142
fminf.....	142
fminl.....	142
fmodf.....	142
fmodl.....	142
fold.....	38
for statement.....	369
fort77.....	38-39
FORTRAN.....	37, 39, 81
fpclassify.....	380
fpos_t.....	390
fprintf.....	391
FP_INFINITE.....	380
FP_NAN.....	380
FP_NORMAL.....	380
FP_SUBNORMAL.....	380
FP_ZERO.....	380
frexpf.....	148
frexpl.....	148
fscanf.....	391
fseeko.....	148
ftello.....	151
ftrylockfile.....	141
FTW.....	200
function	
absolute-value.....	379
exponential.....	378, 382
hyperbolic.....	378, 381
logarithmic.....	378, 382
manipulation.....	379, 385
nearest integer.....	384
power.....	379
quick reference.....	55
remainder.....	385
trigonometric.....	377, 381
unsuffixed.....	389

wide-string numeric conversion	393
funlockfile.....	141
fuser	38
gencat.....	15, 38
general terminal interface	12
generic parameter	389
get	37-39
getaddrinfo.....	146
getbegyx.....	52
getbkgd.....	52
getbkgrnd	52
getcchar.....	52
getch.....	52
getchar_unlocked	155
getconf.....	38
getgrent	128
getgrgid_r	157
getgrnam_r.....	158
gethostent.....	128
getline	156
getlogin_r.....	159
getmaxyx	52
getnetbyaddr.....	129
getnetbyname.....	129
getnetent	129
getnstr	52
getn_wstr.....	52
getopts.....	38
getparyx.....	52
getpmsg.....	159
getprotobyname.....	129
getprotobynumber	129
getprotoent	129
getpwent.....	129
getpwnam_r.....	161
getpwuid_r.....	162
getservbyname.....	129
getservbyport	129
getservent.....	129
getstr	52
getutxent.....	130
getutxid.....	130
getutxline	130
getwin	52
getyx.....	52
get_wch	52
get_wstr.....	52
globfree	164
gmtime_r	165
greatest-width integer type	374
grep	9, 38
grep -E	9

halfdelay	52
hash.....	38
has_colors	52
has_ic	52
has_il	52
hdestroy.....	165
head	38
header	373
quick reference	87
header rules	45
headers	341
hexadecimal constant.....	370
hline.....	52
hline_set.....	52
hsearch.....	165
htons.....	165
hyperbolic function.....	378, 381
hypotf.....	166
hypotl.....	166
l18N.....	15
iconv	15, 38
iconv_close.....	15
iconv_open	15
id.....	38
idcok	52
identifier	371
idlok	52
ilogbf.....	167
ilogbl	167
immedok.....	52
implicit declaration.....	367
inch.....	52
inchnstr.....	52
inchstr.....	52
incomplete array.....	368
inet_ntoa.....	168
inet_pton.....	168
initscr	52
init_color	52
init_pair.....	52
inline	365
innstr.....	52
innwstr.....	52
insch	52
insdelln	52
insertln.....	52
insnstr.....	52
insstr.....	52
instr.....	52
ins_nwstr.....	52
ins_wch	52
ins_wstr	52

integer type.....	373
exact-width.....	373
fastest minimum-width.....	374
greatest-width.....	374
holding object pointer.....	374
minimum-width.....	374
specified-width limit.....	376
interface count.....	2
internationalization.....	15
interprocess communication.....	11
inrflush.....	52
inwstr.....	52
in_wch.....	52
in_wchnstr.....	52
in_wchstr.....	52
ipcrm.....	38
ipcs.....	38
isalnum_l.....	169
isalpha_l.....	170
isblank.....	390
isblank_l.....	171
iscntrl_l.....	171
isdigit_l.....	171
isendwin.....	52
isfinite.....	380
isgraph_l.....	172
isgreater.....	386
isgreaterequal.....	386
isinf.....	380
islower_l.....	173
isnam.....	380
isnormal.....	380
ISO C headers.....	45
ISO C standard.....	363
isprint_l.....	174
ispunct_l.....	174
isspace_l.....	175
isupper_l.....	175
iswalnum_l.....	176
iswalpha_l.....	176
iswblank.....	390
iswblank_l.....	176
iswcntrl_l.....	177
iswctype_l.....	177
iswdigit_l.....	177
iswgraph_l.....	178
iswlower_l.....	178
iswprint_l.....	178
iswpunct_l.....	179
iswspace_l.....	179
iswupper_l.....	179
iswxdigit_l.....	180

isxdigit_l.....	180
is_linetouched	52
is_wintouched.....	52
j1.....	180
jn.....	180
jobs.....	38
join.....	38
jrand48	126
keyname	52
keypad	52
keyword	363
key_name	52
kill	38
killchar	52
killwchar.....	52
l64a.....	102
lchown	9
lcong48.....	126
ld, rationale for omission.....	41
ldexpf	182
ldexpl	182
leaveok	52
lex.....	9, 38-39
lfind.....	187
lgammaf.....	182
lgammal.....	182
library.....	390
line length.....	371
line, rationale for omission.....	41
LINES	52
link	38
linkat	182
lint, rationale for omission.....	41
llabs	181, 390
lldiv	182, 390
llrintf.....	184
llrintl.....	184
llroundf.....	184
llroundl.....	184
ln.....	38
locale	15, 38
localeconv.....	391
localedef	15, 38
localtime_r.....	184
log10f.....	185
log10l.....	185
log1pf.....	185
log1pl.....	185
log2f.....	186
log2l.....	186
logarithmic function	378, 382
logbf.....	186

logbl.....	186
logf.....	185
logger	38
login, rationale for omission.....	41
logl.....	185
logname.....	38
longname.....	52
lorder, rationale for omission	42
lp.....	38
lpstat, rationale for omission.....	42
lrand48	126
lrintf.....	186
lrintl.....	186
lroundf	187
lroundl.....	187
ls.....	38
lstat.....	9, 149
m4	38
macro	376
comparison	386
quick reference	55
mail, rationale for omission.....	42
mailx	14, 38
make.....	38-39
man	38
manipulation function	379, 385
mbsnrtowcs	189
mesg.....	38
meta	52
min/max line length	371
minimum-width integer type.....	374
mkdir.....	38
mkdirat.....	191
mkfifo.....	38
mkfifoat	192
mknod, rationale for omission.....	42
mknodat.....	192
mkstemp.....	191
modff	194
modfl.....	194
more	9, 38
move.....	52
mq_timedreceive	196
mq_timedsend.....	196
mrnd48	126
multiplicative operator.....	367
munlock	193
munlockall.....	193
mv.....	38
mvcur.....	52
mvderwin	52
mvwin	52

name information.....	159
name space.....	10
name space rules.....	45
nanf.....	199
nanl.....	199
napms.....	52
nearbyintf.....	199
nearbyintl.....	199
nearest integer function.....	384
network.....	12
newgrp.....	38
newpad.....	52
news, rationale for omission.....	42
newterm.....	52
newwin.....	52
nextafterf.....	200
nextafterl.....	200
nexttoward.....	200
nexttowardf.....	200
nexttowardl.....	200
nice.....	38
nl.....	9, 38, 52
nl_langinfo_l.....	201
nm.....	38-39
nocbreak.....	52
nodelay.....	52
noecho.....	52
nohup.....	38
nonl.....	52
noqiflush.....	52
noraw.....	52
notimeout.....	52
rand48.....	126
ntohl.....	165
ntohs.....	165
od.....	38
openat.....	201
opendir.....	135
openlog.....	119
open_wmemstream.....	202
optarg.....	160
opterr.....	160
optind.....	160
option group.....	32
option groups.....	6
options policy.....	34
optopt.....	160
overlay.....	52
overwrite.....	52
pack, rationale for omission.....	42
pair_content.....	52
PAIR_NUMBER.....	52

passwd, rationale for omission	42
paste.....	38
patch.....	38
pathchk.....	38
pathconf.....	144
pax.....	9, 38
pcat, rationale for omission.....	42
pechochar.....	52
pecho_wchar	52
pg, rationale for omission	42
pnoutrefresh	52
portability codes	2
posix_spawnattr_init.....	207
posix_spawnattr_setflags	207
posix_spawnattr_setpgroup.....	208
posix_spawnattr_setschedparam	208
posix_spawnattr_setschedpolicy	208
posix_spawnattr_setsigdefault	209
posix_spawnattr_setsigmask.....	209
posix_spawnp.....	206
posix_spawn_file_actions_addopen.....	206
posix_spawn_file_actions_init	207
posix_trace_attr_getcreatetime	210
posix_trace_attr_getgenversion	210
posix_trace_attr_getlogfullpolicy	210
posix_trace_attr_getmaxdatasize.....	211
posix_trace_attr_getmaxsystemeventsz.....	211
posix_trace_attr_getmaxusereventsz.....	211
posix_trace_attr_getname.....	210
posix_trace_attr_getstreamfullpolicy	210
posix_trace_attr_getstreamsize.....	211
posix_trace_attr_init	209
posix_trace_attr_setinherited	210
posix_trace_attr_setlogfullpolicy.....	210
posix_trace_attr_setlogsize.....	211
posix_trace_attr_setmaxdatasize	211
posix_trace_attr_setname	210
posix_trace_attr_setstreamfullpolicy	210
posix_trace_attr_setstreamsize.....	211
posix_trace_create_withlog.....	212
posix_trace_eventid_get_name.....	213
posix_trace_eventid_open.....	212
posix_trace_eventset_del.....	213
posix_trace_eventset_empty.....	213
posix_trace_eventset_fill	213
posix_trace_eventset_ismember	213
posix_trace_eventtypelist_rewind.....	213
posix_trace_flush	212
posix_trace_get_status	214
posix_trace_open	212
posix_trace_rewind.....	212
posix_trace_set_filter	214

posix_trace_shutdown.....	212
posix_trace_stop	215
posix_trace_timedgetnext_event	214
posix_trace_trid_eventid_open.....	213
posix_trace_trygetnext_event.....	214
power function	379
powf.....	215
powl.....	215
pr.....	38
pragma	371
pread	241
predefined identifier.....	366
predefined macro	370
prefresh	52
printf	38, 145, 391
printw.....	52
process	
setting real and effective user IDs.....	255
prof, rationale for omission	42
programming environment.....	9
prs	37-39
ps.....	38
psignal.....	216
pthread_attr_init	217
pthread_attr_setdetachstate.....	217
pthread_attr_setguardsize.....	218
pthread_attr_setinheritsched.....	218
pthread_attr_setschedparam	218
pthread_attr_setschedpolicy	219
pthread_attr_setscope.....	219
pthread_attr_setstack.....	220
pthread_attr_setstacksize.....	220
pthread_barrierattr_init.....	221
pthread_barrierattr_setpshared.....	221
pthread_barrier_init	220
pthread_cleanup_push.....	222
pthread_condattr_init.....	223
pthread_condattr_setclock	224
pthread_condattr_setpshared	224
pthread_cond_init.....	222
pthread_cond_signal.....	222
pthread_cond_wait	223
pthread_mutexattr_init.....	230
pthread_mutexattr_setprioceiling	230
pthread_mutexattr_setprotocol.....	231
pthread_mutexattr_setpshared.....	231
pthread_mutexattr_setrobust.....	232
pthread_mutexattr_settype.....	233
pthread_mutex_init.....	228
PTHREAD_MUTEX_ROBUST.....	232
pthread_mutex_setprioceiling.....	229
PTHREAD_MUTEX_STALLED	232

pthread_mutex_trylock	229
pthread_mutex_unlock	229
pthread_rwlockattr_init	235
pthread_rwlockattr_setpshared	236
pthread_rwlock_init.....	233
pthread_rwlock_tryrdlock.....	234
pthread_rwlock_wrlock	235
pthread_setcanceltype	236
pthread_setconcurrency	225
pthread_setschedparam.....	226
pthread_setspecific	226
pthread_spin_init	237
pthread_spin_trylock	238
pthread_testcancel	236
putchar_unlocked	155
putc_unlocked	155
putp	52
putpmsg.....	239
pututxline	130
putwin	52
pwd.....	38
pwrite.....	297
qalter	38-39
qdel	38-39
qhold	38-39
qiflush	52
qmove.....	38-39
qmsg	38-39
qrerun.....	38-39
qrls	38-39
qselect.....	38-39
qsig.....	38-39
qstat	38-39
qsub	38-39
random	168
rand_r	240
raw.....	52
RCS, rationale for omission	42
read	38
readdir_r	241
readlink.....	9
readlinkat.....	241
realloc.....	391
realpath	9
realtime.....	11, 32
REALTIME	134, 193, 195-197, 247-248, 256-257, 347
realtime threads.....	33
REALTIME THREADS	218-219, 226, 229-231, 237
red, rationale for omission	42
redrawwin	52
refresh	52
regerror.....	243

regexec	243
regfree	243
regular expressions	9
rejected utilities	40
remainder function	385
remainderf	244
remainderl	244
remque	169
remquof	244
remquol	244
renameat	244
renice	38
reserved name	
avoiding	48
resetty	52
reset_prog_mode	52
reset_shell_mode	52
restartterm	52
rintf	246
rintl	246
ripoffline	52
rm	38
rmdel	37-39
rmdir	38
roundf	246
rounding	388
roundl	246
rsh, rationale for omission	42
sact	37-39
savetty	52
scalblnf	246
scalblnl	246
scalbn	246
scalbnf	246
scalbnl	246
scandir	106
scanf	148, 391
scanw	52
SCCS	37
sccs	37-39
sched_get_priority_min	247
scr1	52
scroll	52
scrollok	52
scr_dump	52
scr_init	52
scr_restore	52
scr_set	52
sdb, rationale for omission	42
sdiff, rationale for omission	43
sed	9, 38
seed48	126

select	216
sem_wait	251
setcchar	52
setgrent	128
sethostent	128
setitimer	159
setlogmask	119
setnetent	129
setpriority	161
setprotoent	129
setpwent	129
setrlimit	162
setscreg	52
setservent	129
setstate	168
setupterm	52
setutxent	130
setvbuf	392
set_curterm	52
set_term	52
sh	38
shar, rationale for omission	43
shell command language	13
shl, rationale for omission	43
sigignore	260
signal concepts	10
signbit	380
siggam	182
sigpause	260
sigprocmask	237
sigrelse	260
sigset	260
sigwaitinfo	262
sinf	262
Single UNIX Environment	1
sinh	263
sinhl	263
sinl	262
size, rationale for omission	43
sizeof	367
sleep	38
slk_atroff	52
slk_attron	52
slk_attrset	52
slk_attr_off	52
slk_attr_on	52
slk_attr_set	52
slk_clear	52
slk_color	52
slk_init	52
slk_label	52
slk_noutrefresh	52

slk_refresh	52
slk_restore	52
slk_set	52
slk_touch	52
slk_wset	52
sprintf	145, 392
sockets	12
sort	38
source file inclusion	370
specified-width integer type	376
spell, rationale for omission	43
split	38
sprintf	145, 391
sqrtf	264
sqrtl	264
srand	240
srand48	126
srandom	168
sscanf	148, 391
standard I/O streams	11
standend	52
standout	52
start_color	52
stat	149
statvfs	150
stderr	264
stdout	264
stdscr	52
stpcpy	266
stpncpy	269
strcasecmp_l	264
strcoll_l	265
STREAM	239
STREAMS	11, 135, 159, 169, 354
strerror_l	267
strerror_r	267
strfmon_l	267
strftime	392
strftime_l	268
string literal	367
strings	38
strip	38-39
strncasecmp	264
strncasecmp_l	264
strndup	266
strtod	392
strtof	271, 392
strtok_r	271
strtold	271, 392
strtoll	271, 392
strtoull	272, 392
strtoumax	271

structure member	
incomplete array	368
strxfrm_l	272
stty	38
su, rationale for omission	43
subpad	52
subwin	52
sum, rationale for omission	43
superuser	41
swprintf	153
swscanf	154
symbolic links	13
symlink	9
symlinkat	273
syncok	52
syslog	119
system interfaces	101
tabs	38
tail	38
talk	38
tanf	274
tanhf	274
tanhf	274
tanl	274
tar, rationale for omission	43
tee	38
termattrs	52
terminal interfaces	14, 51
termname	52
term_attrs	52
test	38
tfind	276
tgammaf	277
tgammal	277
threads	12
tigetflag	52
tigetnum	52
tigetstr	52
time	38
timeout	52
timer_gettime	278
timer_settime	278
tiparm	52
tmpnam	393
TMP_MAX	393
token pasting	372
tolower_l	280
touch	38
touchline	52
touchwin	52
toupper_l	280
towctrans_l	280

towlower_l.....	281
toupper_l.....	281
tparam.....	52
tput.....	38
tputs.....	52
tr.....	38
tracing.....	34
TRACING.....	209-215
translation limit.....	371
translation-time arithmetic.....	370
trigonometric function.....	377, 381
true.....	38
truncf.....	281
truncl.....	281
tsearch.....	276
tsort.....	38
tty.....	38
ttynam_e_r.....	282
twalk.....	276
type.....	38, 364
type qualifier.....	364
type-generic math.....	389
typeahead.....	52
tzname.....	282
tzset.....	282
UCN.....	365
UINTn_MAX.....	376
ulimit.....	38
umask.....	38
unalias.....	38
uname.....	38
uncompress.....	38
unctrl.....	52
unexpand.....	38
unget.....	37-39
ungetc.....	393
ungetch.....	52
unget_wch.....	52
uniq.....	38
universal character name.....	365
unlink.....	38
unlinkat.....	283
unpack, rationale for omission.....	43
unsuffixed function	
with C-prefixed counterpart.....	389
without C-prefixed counterpart.....	389
untouchwin.....	52
user ID	
real and effective.....	255
setting real and effective.....	255
use_env.....	52
utilities.....	299

utility	
quick reference	81
utimensat	152
utimes	152
uucp	37-39
uudecode	38
uuencode	38
uustat	37-39
uux	37-39
val	37-39
variable length array	368
variadic macro	372
va_copy	372
vdprintf	285
VFS	356
vfscanf	393
vwscanf	393
vi	9, 38
vidattr	52
vidputs	52
vid_attr	52
vid_puts	52
vline	52
vline_set	52
vprintf	285
vscanf	286
vsprintf	285
vsprintf	285
vsscanf	286
vswprintf	286
vswscanf	286
vwprintf	286
vwscanf	286
vw_printw	52
vw_scanw	52
wait	38
waitpid	287
wall, rationale for omission	43
wattroff	52
watron	52
wattrset	52
wattr_get	52
wattr_off	52
wattr_on	52
wattr_set	52
wbkgd	52
wbkgdset	52
wbkgnd	52
wbkgndset	52
wborder	52
wborder_set	52
wc	38

wclear	52
wclrtoobot	52
wclrtoeol	52
wcolor_set	52
wcpcpy	289
wcpncpy	291
wscasecmp_l	288
wscoll_l	289
wcsncasecmp	288
wcsncasecmp_l	288
wcsnlen	290
wcsnrtoombs	292
wcstof	293
wcstold	293
wcstoll	293
wcstoull	294
wcstoumax	293
wcsxfrm_l	294
wctrans_l	295
wctype_l	295
wcursyncup	52
wdeleteln	52
wechochar	52
wecho_wchar	52
werase	52
wgetbkgrnd	52
what	37-39
who	38
wide-string numeric conversion function	393
winsdelln	52
winsertln	52
wmove	52
wnoutrefresh	52
wordfree	297
wprintf	153
wredrawln	52
wrefresh	52
write	38
wscanf	154
wscr1	52
wsetscrreg	52
wstandend	52
wstandout	52
wsyncdown	52
wsyncup	52
wtimeout	52
wtouchln	52
wunctrl	52
xargs	38
XCU environment	13
XCU options	37

XCURSES	
quick reference	91
XSI headers.....	47
XSI streams.....	34
y1.....	298
yacc.....	8, 38-39
YACC grammar.....	8
yn.....	298
zcat.....	38