

IEEE Standard for Information Technology— Portable Operating System Interface (POSIX™)— Part 1: System Application Program Interface (API)—Amendment 5: Advanced Realtime Extensions [C Language]

Sponsor

Portable Applications Standards Committee
of the
IEEE Computer Society

Approved 30 January 2000

IEEE-SA Standards Board

Abstract: This standard is part of the POSIX series of standards for applications and user interfaces to open systems. It defines the applications interface to system services for synchronization, memory management, time management, and thread management. This standard is stated in terms of its C language binding.

Keywords: API, application portability, C (programming language), data processing, information interchange, open systems, operating system, portable application, POSIX, programming language, realtime, system configuration computer interface

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

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

Copyright © 2000 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published August 2000. Printed in the United States of America.

Print: ISBN 0-7381-1940-7 SH94813
PDF: ISBN 0-7381-1941-5 SS94813

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of the IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, the IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

IEEE is the sole entity that may authorize the use of certification marks, trademarks, or other designations to indicate compliance with the materials set forth herein.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; (978) 750-8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Contents

	PAGE
Introduction	v
Section 1: General	1
1.3 Conformance	1
Section 2: Terminology and General Requirements	3
2.2 Definitions	3
2.2.2 General Terms	3
2.5 Primitive System Data Types	4
2.7 C Language Definitions	5
2.7.3 Headers and Function Prototypes	5
2.8 Numerical Limits	6
2.8.7 Maximum Values	6
2.9 Symbolic Constants	7
2.9.3 Compile-Time Symbolic Constants for Portability Specifications	7
Section 3: Process Primitives	9
3.1 Process Creation and Execution	9
3.1.2 Execute a File	9
3.2 Process Termination	9
3.2.2 Terminate a Process	9
3.3 Signals	10
3.3.8 Synchronously Accept a Signal	10
Section 4: Process Environment	11
4.8 Configurable System Variables	11
4.8.1 Get Configurable System Variables	11
Section 5: Files and Directories	13
5.6 File Characteristics	13
5.6.1 File Characteristics: Header and Data Structure	13
5.6.2 Get File Status	13
5.6.4 Change File Modes	13
Section 6: Input and Output Primitives	15
6.3 File Descriptor Deassignment	15
6.3.1 Close a File	15
6.4 Input and Output	15
6.4.1 Read from a File	15
6.4.2 Write to a File	16
6.5 Control Operations on Files	16
6.5.2 File Control	16
6.5.3 Reposition Read/Write File Offset	16

6.7	Asynchronous Input and Output	16
6.7.8	Wait for Asynchronous I/O Request	16
Section 8:	Language-Specific Services for the C Programming Language . .	17
8.2	C Language Input/Output Functions	17
8.2.2	Open a Stream on a File Descriptor	17
Section 11:	Synchronization	19
11.4	Condition Variables	19
11.4.1	Condition Variable Initialization Attributes	19
11.4.4	Waiting on a Condition	20
11.5	Barriers	21
11.5.1	Barrier Initialization Attributes	21
11.5.2	Initializing and Destroying a Barrier	23
11.5.3	Synchronizing at a Barrier	24
11.6	Reader/Writer Locks	26
11.6.1	Reader/Writer Lock Initialization Attributes	26
11.6.2	Initializing and Destroying a Reader/Writer Lock	28
11.6.3	Applying a Read Lock	29
11.6.4	Applying a Write Lock	32
11.6.5	Unlocking a Reader/Writer Lock	34
11.7	Spin Locks	35
11.7.1	Initializing and Destroying a Spin Lock	35
11.7.2	Locking a Spin Lock	37
11.7.3	Unlocking a Spin Lock	38
Section 12:	Memory Management	41
12.2	Memory Mapping Functions	42
12.2.1	Map Process Addresses to a Memory Object	42
12.2.2	Unmap Previously Mapped Addresses	43
12.2.4	Memory Object Synchronization	44
12.4	Typed Memory Functions	44
12.4.1	Data Definitions	44
12.4.2	Open a Typed Memory Object	44
12.4.3	Find Offset and Length of a Mapped Typed Memory Block . .	47
12.4.4	Query Typed Memory Information	48
Section 14:	Clocks and Timers	51
14.1	Data Definitions for Clocks and Timers	51
14.1.4	Manifest Constants	51
14.2	Clock and Timer Functions	52
14.2.1	Clocks	52
14.2.2	Create a Per-Process Timer	53
14.2.6	High Resolution Sleep with Specifiable Clock	53
Section 18:	Thread Cancellation	57
18.1	Thread Cancellation Overview	57
Annex A (informative)	Bibliography	59
A.4	Other Sources of Information	59
Annex B (informative)	Rationale and Notes	61

B.11 Synchronization	61
B.12 Memory Management	66
B.14 Clocks and Timers	73
B.18 Thread Cancellation	76
Annex F (informative) Portability Considerations	79
F.3 Profiling Considerations	79
Identifier Index	81
Alphabetic Topical Index	83

FIGURES

Figure B-1 – Example of a system with typed memory	67
--	----

TABLES

Table 2-2 – Optional Primitive System Data Types	5
Table 2-11 – Versioned Compile-Time Symbolic Constants	7
Table 4-3 – Optional Configurable System Variables	11

Introduction

(This introduction is not a normative part of 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])

The purpose of this document is to supplement the base standard with interfaces and functionality for applications having realtime requirements or special efficiency requirements in tightly coupled multitasking environments.

This standard will not change the base standard that it amends (including any existing amendments) in such a way that would cause implementations or strictly conforming applications to no longer conform.

This standard defines systems interfaces to support the source portability of applications with realtime requirements. The system interfaces are all extensions of or additions to ISO/IEC 9945-1:1996 [This edition incorporates the base document and extensions for realtime applications (IEEE Std 1003.1b-1993, IEEE Std 1003.1i-1995) and threads (IEEE Std 1003.1c-1995).], as amended by IEEE Std 1003.1d-1999. Although rooted in the culture defined by ISO/IEC 9945-1:1990, they are focused upon the realtime application requirements, and the support of multiple threads of control within a process, which were beyond the scope of ISO/IEC 9945-1:1990. The interfaces included in this standard were the set required to make ISO/IEC 9945-1:1990 efficiently usable to realtime applications or applications running in multiprocessor systems with requirements that were not covered by the realtime or threads extensions specified in IEEE Std 1003.1b-1993, IEEE Std 1003.1c-1995, and IEEE Std 1003.1d-1999. The scope is to take existing realtime or multiprocessor operating system practice and add it to the base standard.

The definition of *realtime* used in defining the scope of this standard is as follows:

Realtime in operating systems: the ability of the operating system to provide a required level of service in a bounded response time.

The key elements of defining the scope are

- (1) Defining a sufficient set of functionality to cover the realtime application program domain in the areas not covered by IEEE Std 1003.1b-1993, IEEE Std 1003.1c-1995, and IEEE Std 1003.1d-1999;
- (2) Defining a sufficient set of functionality to cover efficient synchronization in multiprocessors that allows applications to achieve the performance benefits of such architectures;
- (3) Defining sufficient performance constraints and performance-related functions to allow a realtime application to achieve deterministic response from the system; and
- (4) Specifying changes or additions to improve or complete the definition of the facilities specified in the previous realtime or threads extensions in IEEE Std 1003.1b-1993, IEEE Std 1003.1c-1995, and IEEE Std 1003.1d-1999.

Wherever possible, the requirements of other application environments were included in the interface definition. The specific areas are noted in the scope overviews of each of the interface areas given below.

The specific functional areas included in this standard and their scope include:

- *Synchronization*

New synchronization primitives that allow multiprocessor applications to achieve the performance benefits of their hardware architecture.

- *Memory management*

Memory management allows programs to allocate or access different kinds of physical memory that are present in the system and allows separate application programs to share portions of this memory.

- *Clocks and Timers*

The Monotonic Clock has been added. The effects of setting the time of a clock on other timing services have been specified. Functions have been added to support relative or absolute suspension based upon a clock specified by the application.

This standard has been defined exclusively at the source code level for the C programming language. Although the interfaces will be portable, some of the parameters used by an implementation may have hardware or configuration dependencies.

Related Standards Activities

Activities to extend this standard to address additional requirements are in progress, and similar efforts can be anticipated in the future.

The following areas are under active consideration at this time or are expected to become active in the near future:¹⁾

- (1) Additional system application program interfaces (API) in C language
- (2) Ada language bindings to this standard
- (3) Shell and utility facilities
- (4) Verification testing methods
- (5) Tracing facilities
- (6) Fault tolerance
- (7) Checkpoint/restart facilities
- (8) Resource limiting facilities
- (9) Network interface facilities
- (10) System administration
- (11) Profiles describing application- or user-specific combinations of Open Systems standards
- (12) An overall guide to POSIX-based or -related Open Systems standards and profiles

Extensions are approved as “amendments” or “revisions” to this document, following the IEEE and ISO/IEC procedures.

Approved amendments are published separately until the full document is reprinted and such amendments are incorporated in their proper positions.

If you have interest in participating in the Portable Application Standards Committee (PASC) working groups addressing these issues, please send your name, address, and phone number to the Secretary, IEEE Standards Board, Institute of Electrical and Electronics Engineers, Inc., P.O. Box 1331, 445 Hoes Lane, Piscataway, NJ 08855-1331, and ask to have this information forwarded to the chairperson of the appropriate PASC working group. If you have interest in participating in this work at the international level, contact your ISO/IEC national body.

1) A *Standards Status Report* that lists all current IEEE Computer Society standards projects is available from the IEEE Computer Society, 1730 Massachusetts Avenue NW, Washington, DC 20036-1903; Telephone: +1 202 371-0101; FAX: +1 202 728-9614. Working drafts of POSIX standards under development are also available from this office.

This standard was prepared by the Realtime System Services Working Group, sponsored by the PASC of the IEEE Computer Society. At the time this standard was approved, the membership of the System Services Working Group was as follows:

Portable Application Standards Committee

Chair:	Lowell Johnson
Vice Chair:	Joseph M. Gwinn
Functional Vice Chairs:	Jay Ashford
	Andrew Josey
	Curtis Royster Jr.
Secretary:	Nick Stoughton

Realtime System Services Working Group: Officials

Chair:	Joseph M. Gwinn
	Susan Corwin (until 1995)
Editor:	Michael González
Secretary:	Karen D. Gordon
	Franklin C. Prindle (1996)
	Lee Schemerhorn (to 1995)

Ballot coordinators

Joseph M. Gwinn James T. Oblinger

Technical reviewers

Michael González Karen D. Gordon Franklin C. Prindle

Working Group

Ray Alderman	Bill Gallmeister	Kent Long
Larry Anderson	Michael González	Robert D. Luken
Pierre-Jean Arcos	Karen D. Gordon	James T. Oblinger
Charles R. Arnold	Randy Greene	Offer Pazy
V. Raj Avula	Rick Greer	Franklin C. Prindle
Theodore P. Baker	Joseph M. Gwinn	François Riche
Robert Barned	Steven A. Haaser	Gordon W. Ross
Richard M. Bergman	Barbara Haleen	Curtis Royster, Jr.
Nawaf Bitar	Geoffrey R. Hall	Webb Scales
Steve Brosky	Patrick Hebert	Lee Schermerhorn
David Butenhof	Mary R. Hermann	Lui Sha
Hans Petter Christiansen	David Hughes	Del Swanson
Susan Corwin	Duane Hughes	Barry Traylor
Bill Cox	Michael B. Jones	Stephen R. Wali
Peter Dibble	Steve Kleiman	Andrew E. Wheeler, Jr.
Christoph Eck	Robert Knighten	David Wilner
Michael Feustel	C. Douglass Locke	John Zolnowsky

The following members of the balloting committee voted on this standard:

Phillip R. Acuff	Michael González	David G. Mullens
Alejandro Alonso-Muñoz	Karen D. Gordon	Howard E. Neely, III
Pierre-Jean Arcos	Joseph M. Gwinn	Peter E. Obermayer
Jay Ashford	Steven A. Haaser	James T. Oblinger
Theodore P. Baker	Chris J. Harding	Diane Paul
David Black	Barry Hedquist	Franklin C. Prindle
Shirley Bockstahler-Brandt	Andrew R. Huber	Juan Antonio de la Puente
Mark Brown	Petr Janecek	Helmut Roth
David Butenhof	Lowell G. Johnson	Curtis Royster
James T. Carlo	Michael B. Jones	W. Olin Sibert
Donald Cragun	Andrew Josey	Donn S. Terry
Lee Damico	Michael J. Karels	Mark-Rene Uchida
John S. Davies	Martin J. Kirk	Charlotte Wales
Richard P. Draves	Steven R. Kleiman	Dale G. Wolfe
Philip H. Enslow	C. Douglass Locke	Oren Yuen
Michel P. Gien	Roger J. Martin	Ming De Zhou
	Jerry A. Moore	

The following organizational representatives voted on this standard:

Diane Paul	Andrew Josey
SAE	X/Open Co. Ltd.

When the IEEE-SA Standards Board approved this standard on 30 January 2000, it had the following membership:

Richard J. Holleman, *Chair*
Donald N. Heirman, *Vice Chair*
Judith Gorman, *Secretary*

Satish K. Aggarwal	James H. Gurney	Louis-François Pau
Dennis Bodson	Lowell G. Johnson	Ronald C. Petersen
Mark D. Bowman	Robert Kennelly	Gerald H. Peterson
James T. Carlo	E.G. “Al” Kiener	John B. Posey
Gary R. Engmann	Joseph L. Koepfinger*	Gary S. Robinson
Harold E. Epstein	L. Bruce McClung	Akio Tojo
Jay Forster*	Daleep C. Mohla	Hans E. Weinrich
Ruben D. Garzon	Robert F. Munzner	Donald W. Zispe

*Member emeritus

Also included is the following nonvoting IEEE-SA Standards Board liaison:

Robert E. Hebner

Yvette Ho Sang
IEEE Standards Project Editor

**IEEE Standard for Information Technology—Portable
Operating System Interface (POSIX™)
Part 1: System Application Program Interface
(API)—Amendment 5: Advanced Realtime Extensions
[C Language]**

Section 1: General

1.3 Conformance

1.3.1 Implementation Conformance

⇒ **1.3.1.3 Conforming Implementation Options** *Add (in alphabetical order) to the table of implementation options that warrant requirement by applications or in specifications:*

<code>{_POSIX_BARRIERS}</code>	Barriers option in (2.9.3)
<code>{_POSIX_CLOCK_SELECTION}</code>	Clock Selection option (in 2.9.3)
<code>{_POSIX_MONOTONIC_CLOCK}</code>	Monotonic Clock option (in 2.9.3)
<code>{_POSIX_READER_WRITER_LOCKS}</code>	Reader/Writer Locks option in (2.9.3)
<code>{_POSIX_SPIN_LOCKS}</code>	Spin Locks option (in 2.9.3)
<code>{_POSIX_TYPED_MEMORY_OBJECTS}</code>	Typed Memory Objects option (in 2.9.3)

Section 2: Terminology and General Requirements

2.2 Definitions

2.2.2 General Terms

⇒ **2.2.2 General Terms** *Modify the definition of “memory object” replacing it with the following text:*

2.2.2.63 memory object: Either a file, a shared memory object, or a typed memory object.

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

⇒ **2.2.2 General Terms** *Modify the contents of this subclause to add the following definitions in the correct sorted order [disregarding the subclause numbers shown here].*

2.2.2.150 barrier: A synchronization object that allows multiple threads to synchronize at a particular point in their execution.

2.2.2.151 clock jump: The difference between two successive distinct values of a clock, as observed from the application via one of the “get time” operations.

2.2.2.152 monotonic clock: A clock whose value cannot be set via *clock_settime()* and that cannot have negative clock jumps.

2.2.2.153 reader/writer lock: A synchronization object that gives a group of threads, called “readers,” simultaneous read access to a resource and another group, called “writers,” exclusive write access to the resource. All readers exclude any writers, and a writer excludes all readers and any other writers.

2.2.2.154 spin lock: A synchronization object used to allow multiple threads to serialize their access to shared data.

2.2.2.155 typed memory namespace: A systemwide namespace that contains the names of the typed memory objects present in the system. It is configurable for a given implementation.

2.2.2.156 typed memory object: A combination of a typed memory pool and a typed memory port. The entire contents of the pool shall be accessible from the port. The typed memory object is identified through a name that belongs to the typed memory namespace.

2.2.2.157 typed memory pool: An extent of memory with the same operational characteristics. Typed memory pools may be contained within each other.

2.2.2.158 typed memory port: A hardware access path to one or more typed memory pools.

2.5 Primitive System Data Types

⇒ **2.5 Primitive System Data Types** *Add the following text at the end of the first paragraph, starting “Some data types used by...”*

Support for some primitive data types is dependent on implementation options (see Table 2-2). Where an implementation option is not supported, the primitive data types for that option need not be found in the header `<sys/types.h>`.

⇒ **2.5 Primitive System Data Types** *In the second paragraph, replace “All of the types listed in Table 2-1 ...” by the following:*

“All of the types listed in Table 2-1 and Table 2-2 ...”

⇒ **2.5 Primitive System Data Types** *Add the following datatypes to the list of types for which there are no defined comparison or assignment operators:*

`pthread_barrier_t`, `pthread_barrierattr_t`, `pthread_rwlock_t`,
`pthread_rwlockattr_t`, `pthread_spinlock_t`.

⇒ **2.5 Primitive System Data Types** *Add the following paragraphs after the paragraph starting “There are no defined comparison...”:*

An implementation need not provide the types `pthread_barrier_t` and `pthread_barrierattr_t` unless the Barriers option is supported (see 2.9.3).

An implementation need not provide the types `pthread_rwlock_t` and `pthread_rwlockattr_t` unless the Reader/Writer Locks option is supported (see 2.9.3).

An implementation need not provide the type `pthread_spinlock_t` unless the Spin Locks option is supported (see 2.9.3).

⇒ **2.5 Primitive System Data Types** *Add the following table, and renumber subsequent tables in this Section accordingly:*

Table 2-2 – Optional Primitive System Data Types

Defined Type	Description	Implementation Option
<code>pthread_barrier_t</code>	Used to identify a barrier	Barriers option
<code>pthread_barrierattr_t</code>	Used to define a barrier attributes object	Barriers option
<code>pthread_rwlock_t</code>	Used to identify a reader/writer lock	Reader/Writer Locks option
<code>pthread_rwlockattr_t</code>	Used to define a reader/writer lock attributes object	Reader/Writer Locks option
<code>pthread_spinlock_t</code>	Used to identify a spin lock	Spin Locks option

2.7 C Language Definitions

2.7.3 Headers and Function Prototypes

⇒ **2.7.3 Headers and Function Prototypes** *Add the following text after the sentence “For other functions in this part of ISO/IEC 9945, the prototypes or declarations shall appear in the headers listed below.”:*

Presence of some prototypes or declarations is dependent on implementation options. Where an implementation option is not supported, the prototype or declaration need not be found in the header.

⇒ **2.7.3 Headers and Function Prototypes** *Modify the contents of subclause 2.7.3 to add the following optional functions, at the end of the current list of headers and functions.*

If the Typed Memory Objects option is supported:

```
<sys/mman.h>  posix_typed_mem_open(), posix_mem_offset(),
                posix_typed_mem_get_info()
```

If the Spin Locks option is supported:

```
<pthread.h>   pthread_spin_init(), pthread_spin_destroy(),
                pthread_spin_lock(), pthread_spin_trylock(),
                pthread_spin_unlock()
```

If the Barriers option is supported:

```
<pthread.h>  pthread_barrierattr_init(),
              pthread_barrierattr_destroy(),
              pthread_barrierattr_getpshared(),
              pthread_barrierattr_setpshared(),
              pthread_barrier_init(), pthread_barrier_destroy(),
              pthread_barrier_wait()
```

If the Reader/Writer Locks option is supported:

```
<pthread.h>  pthread_rwlockattr_init(), pthread_rwlockattr_destroy(),
              pthread_rwlockattr_getpshared(),
              pthread_rwlockattr_setpshared(), pthread_rwlock_init(),
              pthread_rwlock_destroy(), pthread_rwlock_rdlock(),
              pthread_rwlock_tryrdlock(),
              pthread_rwlock_timedrdlock(), pthread_rwlock_wrlock(),
              pthread_rwlock_trywrlock(),
              pthread_rwlock_timedwrlock(), pthread_rwlock_unlock()
```

If the Clock Selection option is supported:

```
<time.h>      clock_nanosleep()

<pthread.h>    pthread_condattr_setclock(), pthread_condattr_getclock()
```

2.8 Numerical Limits

2.8.7 Maximum Values

⇒ **2.8.7 Maximum Values** *In Table 2-7a, replace the description of {_POSIX-CLOCKRES_MIN}, currently reading “The CLOCK_REALTIME clock resolution, in nanoseconds”, with the following:*

The resolution of the clocks CLOCK_REALTIME and CLOCK_MONOTONIC (if supported), in nanoseconds

2.9 Symbolic Constants

2.9.3 Compile-Time Symbolic Constants for Portability Specifications

⇒ **2.9.3 Compile-Time Symbolic Constants for Portability Specifications**
Change the first words in the first paragraph, currently saying “The constants in Table 2-10 may be used...” to the following:

The constants in Table 2-10 and Table 2-11 may be used...

⇒ **2.9.3 Compile-Time Symbolic Constants for Portability Specifications**
Add the following sentence at the end of the first paragraph:

If any of the constants in Table 2-11 is defined, it shall be defined with the value shown in that table. This value represents the version of the associated option that is supported by the implementation.

⇒ **2.9.3 Compile-Time Symbolic Constants for Portability Specifications**
Add Table 2-11, shown below, after Table 2-10 renumbering all subsequent tables accordingly.

Table 2-11 – Versioned Compile-Time Symbolic Constants

Name	Value	Description
{_POSIX_BARRIERS}	200001L	If this symbol is defined with the shown value, the implementation supports the Barriers option.
{_POSIX_READER_WRITER_LOCKS}	200001L	If this symbol is defined with the shown value, the implementation supports the Reader/Writer Locks option.
{_POSIX_SPIN_LOCKS}	200001L	If this symbol is defined with the shown value, the implementation supports the Spin Locks option.
{_POSIX_TYPED_MEMORY_OBJECTS}	200001L	If this symbol is defined with the shown value, the implementation supports the Typed Memory Objects option.
{_POSIX_MONOTONIC_CLOCK}	200001L	If this symbol is defined with the shown value, the implementation supports the Monotonic Clock option.
{_POSIX_CLOCK_SELECTION}	200001L	If this symbol is defined with the shown value, the implementation supports the Clock Selection option.

⇒ **2.9.3 Compile-Time Symbolic Constants for Portability Specifications**
Add the following paragraphs:

If the symbol `{_POSIX_BARRIERS}` is defined, then the symbols `{_POSIX_THREADS}` and `{_POSIX_THREAD_SAFE_FUNCTIONS}` shall also be defined. If the symbol `{_POSIX_READER_WRITER_LOCKS}` is defined, then the symbols `{_POSIX_THREADS}` and `{_POSIX_THREAD_SAFE_FUNCTIONS}` shall also be defined. If the symbol `{_POSIX_SPIN_LOCKS}` is defined, then the symbols `{_POSIX_THREADS}` and `{_POSIX_THREAD_SAFE_FUNCTIONS}` shall also be defined.

If the symbol `{_POSIX_MONOTONIC_CLOCK}` is defined, then the symbol `{_POSIX_TIMERS}` shall also be defined.

If the symbol `{_POSIX_CLOCK_SELECTION}` is defined, then the symbol `{_POSIX_TIMERS}` shall also be defined.

Section 3: Process Primitives

3.1 Process Creation and Execution

3.1.2 Execute a File

⇒ **3.1.2.2 Execute a File—Description** *Add the following paragraph after the paragraph starting “If the Memory Mapped Files or Shared Memory Objects option ...”*

If the Typed Memory Objects option is supported, blocks of typed memory that were mapped in the calling process are unmapped, as if *munmap()* was implicitly called to unmap them.

3.2 Process Termination

3.2.2 Terminate a Process

⇒ **3.2.2.2 Terminate a Process—Description** *Add the following list item after item number (11), and renumber the subsequent items accordingly:*

- (12) If the Typed Memory Objects option is supported, blocks of typed memory that were mapped in the calling process are unmapped, as if *munmap()* was implicitly called to unmap them.

3.3 Signals

3.3.8 Synchronously Accept a Signal

⇒ **3.3.8.2 Synchronously Accept a Signal—description** *Add the following text at the end of the paragraph starting “The function `sigtimedwait()` behaves the same as ...”*

If the Monotonic Clock option is supported, the `CLOCK_MONOTONIC` clock shall be used to measure the time interval specified by the *timeout* argument.

Section 4: Process Environment

4.8 Configurable System Variables

4.8.1 Get Configurable System Variables

⇒ **4.8.1.2 Get Configurable System Variables— Description** *Add the following text after the sentence “The implementation shall support all of the variables listed in Table 4-2 and may support others” in the second paragraph:*

Support for some configuration variables is dependent on implementation options (see Table 4-3). Where an implementation option is not supported, the variable need not be supported.

⇒ **4.8.1.2 Get Configurable System Variables— Description** *In the second paragraph, replace the text “The variables in Table 4-2 come from ...” by the following:*

“The variables in Table 4-2 and Table 4-3 come from ...”

⇒ **4.8.1.2 Get Configurable System Variables— Description** *Add the following table:*

Table 4-3 – Optional Configurable System Variables

Variable	<i>name</i> Value
{_POSIX_BARRIERS}	{_SC_BARRIERS}
{_POSIX_READER_WRITER_LOCKS}	{_SC_READER_WRITER_LOCKS}
{_POSIX_SPIN_LOCKS}	{_SC_SPIN_LOCKS}
{_POSIX_TYPED_MEMORY_OBJECTS}	{_SC_TYPED_MEMORY_OBJECTS}
{_POSIX_MONOTONIC_CLOCK}	{_SC_MONOTONIC_CLOCK}
{_POSIX_CLOCK_SELECTION}	{_SC_CLOCK_SELECTION}

Section 5: Files and Directories

5.6 File Characteristics

5.6.1 File Characteristics: Header and Data Structure

⇒ **5.6.1.1 <sys/stat.h> File Types** *Add the following text and macro after `S_TYPEISSSHM`:*

If the Typed Memory Objects option is supported, the implementation may implement typed memory objects as distinct file types, and the following macro shall test whether a file is of the specified type:

`S_TYPEISTMO(buf)` Test macro for a typed memory object

5.6.2 Get File Status

⇒ **5.6.2.2 Get File Status—Description** *Replace the text “If the Shared Memory Objects option is supported and `fildes` references a shared memory object,” by the following:*

If the Shared Memory Objects option is supported and `fildes` references a shared memory object or the Typed Memory Objects option is supported and `fildes` references a typed memory object,

5.6.4 Change File Modes

⇒ **5.6.4.2 Change File Modes—Description** *Add the following paragraph before the paragraph starting “If the calling process does not have appropriate privileges...”:*

If `{_POSIX_TYPED_MEMORY_OBJECTS}` is defined and `fildes` references a typed memory object, the behavior of `fchmod()` is unspecified.

Section 6: Input and Output Primitives

6.3 File Descriptor Deassignment

6.3.1 Close a File

⇒ **6.3.1.2 Close a File—Description** *Replace the paragraph starting “If a memory object remains referenced...” by the following:*

If a shared memory object or a memory mapped file remains referenced at the last close (i.e., a process has it mapped), then the entire contents of the memory object shall persist until the memory object becomes unreferenced. If this is the last close of a shared memory object or a memory mapped file and the close results in the memory object becoming unreferenced, and the memory object has been unlinked, then the memory object shall be removed.

6.4 Input and Output

6.4.1 Read from a File

⇒ **6.4.1.2 Read from a File—Description** *Add the following text at the end of the description:*

If the Typed Memory Objects option is supported:

If *fildev* refers to a typed memory object, the result of the *read()* function is unspecified.

6.4.2 Write to a File

⇒ **6.4.2.2 Write to a File—Description** *Add the following text at the end of the description:*

If the Typed Memory Objects option is supported:

If *fildest* refers to a typed memory object, the result of the *write()* function is unspecified.

6.5 Control Operations on Files

6.5.2 File Control

⇒ **6.5.2.2 File Control—Description** *Add the following text at the end of the description:*

If the Typed Memory Objects option is supported and *fildest* refers to a typed memory object, the result of the *fctl()* function is unspecified.

6.5.3 Reposition Read/Write File Offset

⇒ **6.5.3.2 Reposition Read/Write File Offset—Description** *Add the following text at the end of the description:*

If the Typed Memory Objects option is supported and *fildest* refers to a typed memory object, the result of the *lseek()* function is unspecified.

6.7 Asynchronous Input and Output

6.7.8 Wait for Asynchronous I/O Request

⇒ **6.7.8.2 Wait for Asynchronous I/O Request—Description** *Add the following text at the end of the paragraph starting “If the time interval indicated in ...”:*

If `{_POSIX_MONOTONIC_CLOCK}` is defined, the clock that shall be used to measure this time interval shall be the `CLOCK_MONOTONIC` clock.

Section 8: Language-Specific Services for the C Programming Language

8.2 C Language Input/Output Functions

8.2.2 Open a Stream on a File Descriptor

⇒ **8.2.2.2 Open a Stream on a File Descriptor—Description** *Add the following text at the end of the description:*

If the Typed Memory Objects option is supported and *fildev* refers to a typed memory object, the result of the *fdopen()* function is unspecified.

