

Draft Standard for Information Technology— Portable Operating System Interface (POSIX[®])

Prepared by the Austin Group
(<http://www.opengroup.org/austin/>)

Copyright © 2001 The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA.

Copyright © 2001 The Open Group
Apex Plaza, Forbury Road, Reading, Berkshire RG11AX, UK.

All rights reserved.

Except as permitted below, 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. This is an unapproved draft, subject to change. Permission is hereby granted for Austin Group participants to reproduce this document for purposes of IEEE, the Open Group, and JTC1 standardization activities. Other entities seeking permission to reproduce this document for standardization purposes or other activities must contact the copyright owners for an appropriate license. Use of information contained within this unapproved draft is at your own risk.

Portions of this document are derived with permission from copyrighted material owned by Hewlett-Packard Company, International Business Machines Corporation, Novell Inc., The Open Software Foundation, and Sun Microsystems, Inc.

1

2 **1.1 Scope**

3 IEEE Std 1003.1-200x defines a standard operating system interface and environment, including
4 a command interpreter (or “shell”), and common utility programs to support applications
5 portability at the source code level. It is intended to be used by both applications developers
6 and system implementors.

7 IEEE Std 1003.1-200x comprises four major components (each in an associated volume):

- 8 1. General terms, concepts, and interfaces common to all volumes of IEEE Std 1003.1-200x,
9 including utility conventions and C language header definitions, are included in the Base
10 Definitions volume of IEEE Std 1003.1-200x.
- 11 2. Definitions for system service functions and subroutines, language-specific system
12 services for the C programming language, function issues, including portability, error
13 handling, and error recovery, are included in the System Interfaces volume of
14 IEEE Std 1003.1-200x.
- 15 3. Definitions for a standard source code-level interface to command interpretation services
16 (a “shell”) and common utility programs for application programs are included in the
17 Shell and Utilities volume of IEEE Std 1003.1-200x.
- 18 4. Extended rationale that did not fit well into the rest of the document structure, containing
19 historical information concerning the contents of IEEE Std 1003.1-200x and why features
20 were included or discarded by the standard developers, is included in the Rationale
21 (Informative) volume of IEEE Std 1003.1-200x.

22 The following areas are outside of the scope of IEEE Std 1003.1-200x:

- 23 • Graphics interfaces
- 24 • Database management system interfaces
- 25 • Record I/O considerations
- 26 • Object or binary code portability
- 27 • System configuration and resource availability

28 IEEE Std 1003.1-200x describes the external characteristics and facilities that are of importance to
29 applications developers, rather than the internal construction techniques employed to achieve
30 these capabilities. Special emphasis is placed on those functions and facilities that are needed in
31 a wide variety of commercial applications.

32 The facilities provided in IEEE Std 1003.1-200x are drawn from the following base documents:

- 33 • IEEE Std 1003.1-1996 (POSIX-1) (incorporating IEEE Stds. 1003.1-1990, 1003.1b-1993,
34 1003.1c-1995, and 1003.1i-1995)
- 35 • The following amendments to the POSIX.1-1990 standard:
 - 36 — IEEE P1003.1a draft standard (Additional System Services)
 - 37 — IEEE Std 1003.1d-1999 (Additional Realtime Extensions)

- 38 — IEEE Std 1003.1g-2000 (Protocol-Independent Interfaces (PII))
- 39 — IEEE Std 1003.1j-2000 (Advanced Realtime Extensions)
- 40 — IEEE Std 1003.1q-2000 (Tracing)
- 41 • IEEE Std 1003.2-1992 (POSIX-2) (includes IEEE Std 1003.2a-1992)
- 42 • The following amendments to the ISO POSIX-2: 1993 standard:
- 43 — IEEE P1003.2b draft standard (Additional Utilities)
- 44 — IEEE Std 1003.2d-1994 (Batch Environment)
- 45 • Open Group Technical Standard, February 1997, System Interface Definitions, Issue 5 (XBD5)
- 46 (ISBN: 1-85912-186-1, C605)
- 47 • Open Group Technical Standard, February 1997, Commands and Utilities, Issue 5 (XCU5)
- 48 (ISBN: 1-85912-191-8, C604)
- 49 • Open Group Technical Standard, February 1997, System Interfaces and Headers, Issue 5
- 50 (XSH5) (in 2 Volumes) (ISBN: 1-85912-181-0, C606)
- 51 **Note:** XBD5, XCU5, and XSH5 are collectively referred to as the *Base Specifications*.
- 52 • Open Group Technical Standard, January 2000, Networking Services, Issue 5.2 (XNS5.2)
- 53 (ISBN: 1-85912-241-8, C808)
- 54 • ISO/IEC 9899: 1999, Programming Languages — C.

55 IEEE Std 1003.1-200x uses the *Base Specifications* as its organizational basis and adds the
56 following additional functionality to them drawn from the base documents above:

- 57 • Normative text from the ISO POSIX-1: 1996 standard and the ISO POSIX-2: 1993 standard not
- 58 included in the *Base Specifications*
- 59 • The amendments to the POSIX.1-1990 standard and the ISO POSIX-2: 1993 standard listed
- 60 above, except for parts of IEEE Std 1003.1g-2000
- 61 • Portability Considerations
- 62 • Additional rationale and notes

63 The following features, marked legacy or obsolescent in the base documents, are not carried
64 forward into IEEE Std 1003.1-200x. Other features from the base documents marked legacy or
65 obsolescent are carried forward unless otherwise noted.

66 From XSH5, the following legacy interfaces, headers, and external variables are not carried
67 forward:

68 *advance()*, *brk()*, *chroot()*, *compile()*, *cuserid()*, *gamma()*, *getdtablesize()*, *getpagesize()*, *getpass()*,
69 *getw()*, *putw()*, *re_comp()*, *re_exec()*, *regcmp()*, *sbrk()*, *sigstack()*, *step()*, *wait3()*, **<re_comp.h>**,
70 **<regexp.h>**, **<varargs.h>**, *loc1*, *__loc1*, *loc2*, *locs*

71 From XCU5, the following legacy utilities are not carried forward:

72 *calendar*, *cancel*, *cc*, *col*, *cpio*, *cu*, *dircmp*, *dis*, *egrep*, *fgrep*, *line*, *lint*, *lpstat*, *mail*, *pack*, *pcat*, *pg*, *spell*,
73 *sum*, *tar*, *unpack*, *uulog*, *uname*, *uupick*, *uuto*

74 In addition, legacy features within non-legacy reference pages (for example, headers) are not
75 carried forward.

76 From the ISO POSIX-1:1996 standard, the following obsolescent features are not carried
77 forward:

- 78 Page 112, CLK_TCK
 79 Page 197 *tcgetattr()* rate returned option
- 80 From the ISO POSIX-2:1993 standard, obsolescent features within the following pages are not
 81 carried forward:
- 82 Page 75, zero-length prefix within PATH
 83 Page 156, 159 *set*
 84 Page 178, *awk*, use of no argument and no parentheses with length
 85 Page 259, *ed*
 86 Page 272, *env*
 87 Page 282, *find -perm[-]onum*
 88 Page 295-296, *egrep*
 89 Page 299-300, *head*
 90 Page 305-306, *join*
 91 Page 309-310, *kill*
 92 Page 431-433, 435-436, *sort*
 93 Page 444-445, *tail*
 94 Page 453, 455-456, *touch*
 95 Page 464-465, *tty*
 96 Page 472, *uniq*
 97 Page 515-516, *ex*
 98 Page 542-543, *expand*
 99 Page 563-565, *more*
 100 Page 574-576, *newgrp*
 101 Page 578, *nice*
 102 Page 594-596, *renice*
 103 Page 597-598, *split*
 104 Page 600-601, *strings*
 105 Page 624-625, *vi*
 106 Page 693, *lex*
- 107 The *c89* utility (which specified a compiler for the C Language specified by the
 108 ISO/IEC 9899:1990 standard) has been replaced by a *c99* utility (which specifies a compiler for
 109 the C Language specified by the ISO/IEC 9899:1999 standard).
- 110 From XSH5, text marked OH (Optional Header) has been reviewed on a case-by-case basis and |
 111 removed where appropriate. The XCU5 text marked OF (Output Format Incompletely Specified) |
 112 and UN (Possibly Unsupportable Feature) has been reviewed on a case-by-case basis and |
 113 removed where appropriate |
- 114 For the networking interfaces, the base document is the XNS, Issue 5.2 specification. The
 115 following parts of the XNS, Issue 5.2 specification are out of scope and not included in
 116 IEEE Std 1003.1-200x:
- 117 • Part 3 (XTI)
 - 118 • Part 4 (Appendixes)
- 119 Since there is much duplication between the XNS, Issue 5.2 specification and
 120 IEEE Std 1003.1g-2000, material only from the following sections of IEEE Std 1003.1g-2000 has
 121 been included:
- 122 • General terms related to sockets (2.2.2)
 - 123 • Socket concepts (5.1 through 5.3, inclusive)

- 124 • The *pselect()* function (6.2.2.1 and 6.2.3)
- 125 • The `<sys/select.h>` header (6.2)

126 Emphasis is placed on standardizing existing practice for existing users, with changes and
 127 additions limited to correcting deficiencies in the following areas:

- 128 • Issues raised by IEEE or ISO/IEC Interpretations against IEEE Std 1003.1 and IEEE Std 1003.2
- 129 • Issues raised in corrigenda for the *Base Specifications* and working group resolutions from The
 130 Open Group
- 131 • Corrigenda and resolutions passed by The Open Group for the XNS, Issue 5.2 specification
- 132 • Changes to make the text self-consistent with the additional material merged
- 133 • A reorganization of the options in order to facilitate profiling, both for smaller profiles such
 134 as IEEE Std 1003.13, and larger profiles such as the Single UNIX Specification
- 135 • Alignment with the ISO/IEC 9899: 1999 standard

136 1.2 Conformance

137 Conformance requirements for IEEE Std 1003.1-200x are defined in Chapter 2 (on page 15).

138 1.3 Normative References

139 The following standards contain provisions which, through references in IEEE Std 1003.1-200x,
 140 constitute provisions of IEEE Std 1003.1-200x. At the time of publication, the editions indicated
 141 were valid. All standards are subject to revision, and parties to agreements based on
 142 IEEE Std 1003.1-200x are encouraged to investigate the possibility of applying the most recent
 143 editions of the standards listed below. Members of IEC and ISO maintain registers of currently
 144 valid International Standards.

145 ANS X3.9-1978

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

148 ISO/IEC 646: 1991

149 ISO/IEC 646: 1991, Information Processing — ISO 7-bit Coded Character Set for Information
 150 Interchange.²

151 The reference version of the standard contains 95 graphic characters, which are identical to
 152 the graphic characters defined in the ASCII coded character set.

153 ISO 4217: 1995

154 ISO 4217: 1995, Codes for the Representation of Currencies and Funds. |

155

156 1. ANSI documents can be obtained from the Sales Department, American National Standards Institute, 1430 Broadway, New
 157 York, NY 10018, U.S.A.

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

160	ISO 8601:2000	
161	ISO 8601:2000, Data Elements and Interchange Formats — Information Interchange —	
162	Representation of Dates and Times.	
163	ISO C (1999)	
164	ISO/IEC 9899:1999, Programming Languages — C, including Technical Corrigendum No. 1.	
165	ISO/IEC 10646-1:2000	
166	ISO/IEC 10646-1:2000, Information Technology — Universal Multiple-Octet Coded	
167	Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.	

168 1.4 Terminology

169 For the purposes of IEEE Std 1003.1-200x, the following terminology definitions apply:

170 **can**

171 Describes a permissible optional feature or behavior available to the user or application. The
172 feature or behavior is mandatory for an implementation that conforms to
173 IEEE Std 1003.1-200x. An application can rely on the existence of the feature or behavior.

174 **implementation-defined**

175 Describes a value or behavior that is not defined by IEEE Std 1003.1-200x but is selected by
176 an implementor. The value or behavior may vary among implementations that conform to
177 IEEE Std 1003.1-200x. An application should not rely on the existence of the value or
178 behavior. An application that relies on such a value or behavior cannot be assured to be
179 portable across conforming implementations.

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

182 **legacy**

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

187 **may**

188 Describes a feature or behavior that is optional for an implementation that conforms to
189 IEEE Std 1003.1-200x. An application should not rely on the existence of the feature or
190 behavior. An application that relies on such a feature or behavior cannot be assured to be
191 portable across conforming implementations.

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

193 **shall**

194 For an implementation that conforms to IEEE Std 1003.1-200x, describes a feature or
195 behavior that is mandatory. An application can rely on the existence of the feature or
196 behavior.

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

198 **should**

199 For an implementation that conforms to IEEE Std 1003.1-200x, describes a feature or
200 behavior that is recommended but not mandatory. An application should not rely on the
201 existence of the feature or behavior. An application that relies on such a feature or behavior
202 cannot be assured to be portable across conforming implementations.

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

205 **undefined**

206 Describes the nature of a value or behavior not defined by IEEE Std 1003.1-200x which
207 results from use of an invalid program construct or invalid data input.

208 The value or behavior may vary among implementations that conform to
209 IEEE Std 1003.1-200x. An application should not rely on the existence or validity of the
210 value or behavior. An application that relies on any particular value or behavior cannot be
211 assured to be portable across conforming implementations.

212 **unspecified**

213 Describes the nature of a value or behavior not specified by IEEE Std 1003.1-200x which
214 results from use of a valid program construct or valid data input.

215 The value or behavior may vary among implementations that conform to
216 IEEE Std 1003.1-200x. An application should not rely on the existence or validity of the
217 value or behavior. An application that relies on any particular value or behavior cannot be
218 assured to be portable across conforming implementations.

219 **1.5 Portability**

220 Some of the utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x and functions in
221 the System Interfaces volume of IEEE Std 1003.1-200x describe functionality that might not be
222 fully portable to systems meeting the requirements for POSIX conformance (see the Base
223 Definitions volume of IEEE Std 1003.1-200x, Chapter 2, Conformance).

224 Where optional, enhanced, or reduced functionality is specified, the text is shaded and a code in
225 the margin identifies the nature of the option, extension, or warning (see Section 1.5.1). For
226 maximum portability, an application should avoid such functionality.

227 Unless the primary task of a utility is to produce textual material on its standard output,
228 application developers should not rely on the format or content of any such material that may be
229 produced. Where the primary task *is* to provide such material, but the output format is
230 incompletely specified, the description is marked with the OF margin code and shading.
231 Application developers are warned not to expect that the output of such an interface on one
232 system is any guide to its behavior on another system.

233 **1.5.1 Codes**

234 The codes and their meanings are as follows. See also Section 1.5.2 (on page 14).

235 ADV **Advisory Information**

236 The functionality described is optional. The functionality described is also an extension to the
237 ISO C standard.

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

241 AIO **Asynchronous Input and Output**

242 The functionality described is optional. The functionality described is also an extension to the
243 ISO C standard.

244 Where applicable, functions are marked with the AIO margin legend in the SYNOPSIS section.
245 Where additional semantics apply to a function, the material is identified by use of the AIO

246		margin legend.
247	BAR	Barriers
248		The functionality described is optional. The functionality described is also an extension to the
249		ISO C standard.
250		Where applicable, functions are marked with the BAR margin legend in the SYNOPSIS section.
251		Where additional semantics apply to a function, the material is identified by use of the BAR
252		margin legend.
253	BE	Batch Environment Services and Utilities
254		The functionality described is optional.
255		Where applicable, utilities are marked with the BE margin legend in the SYNOPSIS section.
256		Where additional semantics apply to a utility, the material is identified by use of the BE margin
257		legend.
258	CD	C-Language Development Utilities
259		The functionality described is optional.
260		Where applicable, utilities are marked with the CD margin legend in the SYNOPSIS section.
261		Where additional semantics apply to a utility, the material is identified by use of the CD margin
262		legend.
263	CPT	Process CPU-Time Clocks
264		The functionality described is optional. The functionality described is also an extension to the
265		ISO C standard.
266		Where applicable, functions are marked with the CPT margin legend in the SYNOPSIS section.
267		Where additional semantics apply to a function, the material is identified by use of the CPT
268		margin legend.
269	CS	Clock Selection
270		The functionality described is optional. The functionality described is also an extension to the
271		ISO C standard.
272		Where applicable, functions are marked with the CS margin legend in the SYNOPSIS section.
273		Where additional semantics apply to a function, the material is identified by use of the CS
274		margin legend.
275	CX	Extension to the ISO C standard
276		The functionality described is an extension to the ISO C standard. Application writers may make
277		use of an extension as it is supported on all IEEE Std 1003.1-200x-conforming systems.
278		With each function or header from the ISO C standard, a statement to the effect that “any
279		conflict is unintentional” is included. That is intended to refer to a direct conflict.
280		IEEE Std 1003.1-200x acts in part as a profile of the ISO C standard, and it may choose to further
281		constrain behaviors allowed to vary by the ISO C standard. Such limitations are not considered
282		conflicts.
283	FD	FORTTRAN Development Utilities
284		The functionality described is optional.
285		Where applicable, utilities are marked with the FD margin legend in the SYNOPSIS section.
286		Where additional semantics apply to a utility, the material is identified by use of the FD margin
287		legend.
288	FR	FORTTRAN Runtime Utilities
289		The functionality described is optional.

290 Where applicable, utilities are marked with the FR margin legend in the SYNOPSIS section.
291 Where additional semantics apply to a utility, the material is identified by use of the FR margin
292 legend.

293 FSC **File Synchronization**

294 The functionality described is optional. The functionality described is also an extension to the
295 ISO C standard.

296 Where applicable, functions are marked with the FSC margin legend in the SYNOPSIS section.
297 Where additional semantics apply to a function, the material is identified by use of the FSC
298 margin legend.

299 IP6 **IPV6**

300 The functionality described is optional. The functionality described is also an extension to the
301 ISO C standard.

302 Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section.
303 Where additional semantics apply to a function, the material is identified by use of the IP6
304 margin legend.

305 MC1 **Advisory Information and either Memory Mapped Files or Shared Memory Objects**

306 The functionality described is optional. The functionality described is also an extension to the
307 ISO C standard.

308 This is a shorthand notation for combinations of multiple option codes.

309 Where applicable, functions are marked with the MC1 margin legend in the SYNOPSIS section.
310 Where additional semantics apply to a function, the material is identified by use of the MC1
311 margin legend.

312 Refer to Section 1.5.2 (on page 14).

313 MC2 **Memory Mapped Files, Shared Memory Objects, or Memory Protection**

314 The functionality described is optional. The functionality described is also an extension to the
315 ISO C standard.

316 This is a shorthand notation for combinations of multiple option codes.

317 Where applicable, functions are marked with the MC2 margin legend in the SYNOPSIS section.
318 Where additional semantics apply to a function, the material is identified by use of the MC2
319 margin legend.

320 Refer to Section 1.5.2 (on page 14).

321 MF **Memory Mapped Files**

322 The functionality described is optional. The functionality described is also an extension to the
323 ISO C standard.

324 Where applicable, functions are marked with the MF margin legend in the SYNOPSIS section.
325 Where additional semantics apply to a function, the material is identified by use of the MF
326 margin legend.

327 ML **Process Memory Locking**

328 The functionality described is optional. The functionality described is also an extension to the
329 ISO C standard.

330 Where applicable, functions are marked with the ML margin legend in the SYNOPSIS section.
331 Where additional semantics apply to a function, the material is identified by use of the ML
332 margin legend.

333	MLR	Range Memory Locking
334		The functionality described is optional. The functionality described is also an extension to the
335		ISO C standard.
336		Where applicable, functions are marked with the MLR margin legend in the SYNOPSIS section.
337		Where additional semantics apply to a function, the material is identified by use of the MLR
338		margin legend.
339	MON	Monotonic Clock
340		The functionality described is optional. The functionality described is also an extension to the
341		ISO C standard.
342		Where applicable, functions are marked with the MON margin legend in the SYNOPSIS section.
343		Where additional semantics apply to a function, the material is identified by use of the MON
344		margin legend.
345	MPR	Memory Protection
346		The functionality described is optional. The functionality described is also an extension to the
347		ISO C standard.
348		Where applicable, functions are marked with the MPR margin legend in the SYNOPSIS section.
349		Where additional semantics apply to a function, the material is identified by use of the MPR
350		margin legend.
351	MSG	Message Passing
352		The functionality described is optional. The functionality described is also an extension to the
353		ISO C standard.
354		Where applicable, functions are marked with the MSG margin legend in the SYNOPSIS section.
355		Where additional semantics apply to a function, the material is identified by use of the MSG
356		margin legend.
357	MX	IEC 60559 Floating-Point Option
358		The functionality described is optional. The functionality described is also an extension to the
359		ISO C standard.
360		Where applicable, functions are marked with the MX margin legend in the SYNOPSIS section.
361		Where additional semantics apply to a function, the material is identified by use of the MX
362		margin legend.
363	OB	Obsolescent
364		The functionality described may be withdrawn in a future version of this volume of
365		IEEE Std 1003.1-200x. Strictly Conforming POSIX Applications and Strictly Conforming XSI
366		Applications shall not use obsolescent features.
367	OF	Output Format Incompletely Specified
368		The functionality described is an XSI extension. The format of the output produced by the utility
369		is not fully specified. It is therefore not possible to post-process this output in a consistent
370		fashion. Typical problems include unknown length of strings and unspecified field delimiters.
371	OH	Optional Header
372		In the SYNOPSIS section of some interfaces in the System Interfaces volume of
373		IEEE Std 1003.1-200x an included header is marked as in the following example:
374	OH	<code>#include <sys/types.h></code>
375		<code>#include <grp.h></code>
376		<code>struct group *getgrnam(const char *name);</code>

377 This indicates that the marked header is not required on XSI-conformant systems.

378 PIO **Prioritized Input and Output**
379 The functionality described is optional. The functionality described is also an extension to the
380 ISO C standard.

381 Where applicable, functions are marked with the PIO margin legend in the SYNOPSIS section.
382 Where additional semantics apply to a function, the material is identified by use of the PIO
383 margin legend.

384 PS **Process Scheduling**
385 The functionality described is optional. The functionality described is also an extension to the
386 ISO C standard.

387 Where applicable, functions are marked with the PS margin legend in the SYNOPSIS section.
388 Where additional semantics apply to a function, the material is identified by use of the PS
389 margin legend.

390 RS **Raw Sockets**
391 The functionality described is optional. The functionality described is also an extension to the
392 ISO C standard.

393 Where applicable, functions are marked with the RS margin legend in the SYNOPSIS section.
394 Where additional semantics apply to a function, the material is identified by use of the RS
395 margin legend.

396 RTS **Realtime Signals Extension**
397 The functionality described is optional. The functionality described is also an extension to the
398 ISO C standard.

399 Where applicable, functions are marked with the RTS margin legend in the SYNOPSIS section.
400 Where additional semantics apply to a function, the material is identified by use of the RTS
401 margin legend.

402 SD **Software Development Utilities**
403 The functionality described is optional.

404 Where applicable, utilities are marked with the SD margin legend in the SYNOPSIS section.
405 Where additional semantics apply to a utility, the material is identified by use of the SD margin
406 legend.

407 SEM **Semaphores**
408 The functionality described is optional. The functionality described is also an extension to the
409 ISO C standard.

410 Where applicable, functions are marked with the SEM margin legend in the SYNOPSIS section.
411 Where additional semantics apply to a function, the material is identified by use of the SEM
412 margin legend.

413 SHM **Shared Memory Objects**
414 The functionality described is optional. The functionality described is also an extension to the
415 ISO C standard.

416 Where applicable, functions are marked with the SHM margin legend in the SYNOPSIS section.
417 Where additional semantics apply to a function, the material is identified by use of the SHM
418 margin legend.

419 SIO **Synchronized Input and Output**
420 The functionality described is optional. The functionality described is also an extension to the
421 ISO C standard.

422 Where applicable, functions are marked with the SIO margin legend in the SYNOPSIS section.
423 Where additional semantics apply to a function, the material is identified by use of the SIO
424 margin legend.

425 SPI **Spin Locks**
426 The functionality described is optional. The functionality described is also an extension to the
427 ISO C standard.

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

431 SPN **Spawn**
432 The functionality described is optional. The functionality described is also an extension to the
433 ISO C standard.

434 Where applicable, functions are marked with the SPN margin legend in the SYNOPSIS section.
435 Where additional semantics apply to a function, the material is identified by use of the SPN
436 margin legend.

437 SS **Process Sporadic Server**
438 The functionality described is optional. The functionality described is also an extension to the
439 ISO C standard.

440 Where applicable, functions are marked with the SS margin legend in the SYNOPSIS section.
441 Where additional semantics apply to a function, the material is identified by use of the SS
442 margin legend.

443 TCT **Thread CPU-Time Clocks**
444 The functionality described is optional. The functionality described is also an extension to the
445 ISO C standard.

446 Where applicable, functions are marked with the TCT margin legend in the SYNOPSIS section.
447 Where additional semantics apply to a function, the material is identified by use of the TCT
448 margin legend.

449 TEF **Trace Event Filter**
450 The functionality described is optional. The functionality described is also an extension to the
451 ISO C standard.

452 Where applicable, functions are marked with the TEF margin legend in the SYNOPSIS section.
453 Where additional semantics apply to a function, the material is identified by use of the TEF
454 margin legend.

455 THR **Threads**
456 The functionality described is optional. The functionality described is also an extension to the
457 ISO C standard.

458 Where applicable, functions are marked with the THR margin legend in the SYNOPSIS section.
459 Where additional semantics apply to a function, the material is identified by use of the THR
460 margin legend.

461 TMO **Timeouts**
462 The functionality described is optional. The functionality described is also an extension to the
463 ISO C standard.

464 Where applicable, functions are marked with the TMO margin legend in the SYNOPSIS section.
465 Where additional semantics apply to a function, the material is identified by use of the TMO
466 margin legend.

467	TMR	Timers
468		The functionality described is optional. The functionality described is also an extension to the
469		ISO C standard.
470		Where applicable, functions are marked with the TMR margin legend in the SYNOPSIS section.
471		Where additional semantics apply to a function, the material is identified by use of the TMR
472		margin legend.
473	TPI	Thread Priority Inheritance
474		The functionality described is optional. The functionality described is also an extension to the
475		ISO C standard.
476		Where applicable, functions are marked with the TPI margin legend in the SYNOPSIS section.
477		Where additional semantics apply to a function, the material is identified by use of the TPI
478		margin legend.
479	TPP	Thread Priority Protection
480		The functionality described is optional. The functionality described is also an extension to the
481		ISO C standard.
482		Where applicable, functions are marked with the TPP margin legend in the SYNOPSIS section.
483		Where additional semantics apply to a function, the material is identified by use of the TPP
484		margin legend.
485	TPS	Thread Execution Scheduling
486		The functionality described is optional. The functionality described is also an extension to the
487		ISO C standard.
488		Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
489		Where additional semantics apply to a function, the material is identified by use of the TPS
490		margin legend.
491	TRC	Trace
492		The functionality described is optional. The functionality described is also an extension to the
493		ISO C standard.
494		Where applicable, functions are marked with the TRC margin legend in the SYNOPSIS section.
495		Where additional semantics apply to a function, the material is identified by use of the TRC
496		margin legend.
497	TRI	Trace Inherit
498		The functionality described is optional. The functionality described is also an extension to the
499		ISO C standard.
500		Where applicable, functions are marked with the TRI margin legend in the SYNOPSIS section.
501		Where additional semantics apply to a function, the material is identified by use of the TRI
502		margin legend.
503	TRL	Trace Log
504		The functionality described is optional. The functionality described is also an extension to the
505		ISO C standard.
506		Where applicable, functions are marked with the TRL margin legend in the SYNOPSIS section.
507		Where additional semantics apply to a function, the material is identified by use of the TRL
508		margin legend.
509	TSA	Thread Stack Address Attribute
510		The functionality described is optional. The functionality described is also an extension to the
511		ISO C standard.

512 Where applicable, functions are marked with the TSA margin legend for the SYNOPSIS section.
513 Where additional semantics apply to a function, the material is identified by use of the TSA
514 margin legend.

515 TSF **Thread-Safe Functions**

516 The functionality described is optional. The functionality described is also an extension to the
517 ISO C standard.

518 Where applicable, functions are marked with the TSF margin legend in the SYNOPSIS section.
519 Where additional semantics apply to a function, the material is identified by use of the TSF
520 margin legend.

521 TSH **Thread Process-Shared Synchronization**

522 The functionality described is optional. The functionality described is also an extension to the
523 ISO C standard.

524 Where applicable, functions are marked with the TSH margin legend in the SYNOPSIS section.
525 Where additional semantics apply to a function, the material is identified by use of the TSH
526 margin legend.

527 TSP **Thread Sporadic Server**

528 The functionality described is optional. The functionality described is also an extension to the
529 ISO C standard.

530 Where applicable, functions are marked with the TSP margin legend in the SYNOPSIS section.
531 Where additional semantics apply to a function, the material is identified by use of the TSP
532 margin legend.

533 TSS **Thread Stack Address Size**

534 The functionality described is optional. The functionality described is also an extension to the
535 ISO C standard.

536 Where applicable, functions are marked with the TSS margin legend in the SYNOPSIS section.
537 Where additional semantics apply to a function, the material is identified by use of the TSS
538 margin legend.

539 TYM **Typed Memory Objects**

540 The functionality described is optional. The functionality described is also an extension to the
541 ISO C standard.

542 Where applicable, functions are marked with the TYM margin legend in the SYNOPSIS section.
543 Where additional semantics apply to a function, the material is identified by use of the TYM
544 margin legend.

545 UP **User Portability Utilities**

546 The functionality described is optional.

547 Where applicable, utilities are marked with the UP margin legend in the SYNOPSIS section.
548 Where additional semantics apply to a utility, the material is identified by use of the UP margin
549 legend.

550 XSI **Extension**

551 The functionality described is an XSI extension. Functionality marked XSI is also an extension to
552 the ISO C standard. Application writers may confidently make use of an extension on all
553 systems supporting the X/Open System Interfaces Extension.

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

556 XSR **XSI STREAMS**
557 The functionality described is optional. The functionality described is also an extension to the
558 ISO C standard.

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

562 1.5.2 Margin Code Notation

563 Some of the functionality described in IEEE Std 1003.1-200x depends on support of more than
564 one option, or independently may depend on several options. The following notation for margin
565 codes is used to denote the following cases.

566 A Feature Dependent on One or Two Options

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

568 MF This feature requires support for only the Memory Mapped Files option.

569 MF SHM This feature requires support for both the Memory Mapped Files and the Shared Memory
570 Objects options; that is, an application which uses this feature is portable only between
571 implementations that provide both options.

572 A Feature Dependent on Either of the Options Denoted

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

574 MF|SHM This feature is dependent on support for either the Memory Mapped Files option or the Shared
575 Memory Objects option; that is, an application which uses this feature is portable between
576 implementations that provide any (or all) of the options.

577 A Feature Dependent on More than Two Options

578 The following shorthand notations are used:

579 MC1 The MC1 margin code is shorthand for ADV (MF|SHM). Features which are shaded with this
580 margin code require support of the Advisory Information option and either the Memory
581 Mapped Files or Shared Memory Objects option.

582 MC2 The MC2 margin code is shorthand for MF|SHM|MPR. Features which are shaded with this
583 margin code require support of either the Memory Mapped Files, Shared Memory Objects, or
584 Memory Protection options.

585 Large Sections Dependent on an Option

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

588 TRC This section describes extensions to support tracing of user applications. This functionality is
589 dependent on support of the Trace option (and the rest of this section is not further shaded for
590 this option).

591

592 **2.1 Implementation Conformance**

593 **2.1.1 Requirements**

594 A *conforming implementation* shall meet all of the following criteria:

- 595 1. The system shall support all utilities, functions, and facilities defined within
596 IEEE Std 1003.1-200x that are required for POSIX conformance (see Section 2.1.3 (on page
597 16)). These interfaces shall support the functional behavior described herein.
- 598 2. The system may support one or more options as described under Section 2.1.5 (on page
599 20). When an implementation claims that an option is supported, all of its constituent
600 parts shall be provided.
- 601 3. The system may support the X/Open System Interface Extension (XSI) as described under
602 Section 2.1.4 (on page 19).
- 603 4. The system may provide additional utilities, functions, or facilities not required by
604 IEEE Std 1003.1-200x. Non-standard extensions of the utilities, functions, or facilities
605 specified in IEEE Std 1003.1-200x should be identified as such in the system
606 documentation. Non-standard extensions, when used, may change the behavior of utilities,
607 functions, or facilities defined by IEEE Std 1003.1-200x. The conformance document shall
608 define an environment in which an application can be run with the behavior specified by
609 IEEE Std 1003.1-200x. In no case shall such an environment require modification of a
610 Strictly Conforming POSIX Application (see Section 2.2.1 (on page 29)).

611 **2.1.2 Documentation**

612 A conformance document with the following information shall be available for an
613 implementation claiming conformance to IEEE Std 1003.1-200x. The conformance document
614 shall have the same structure as IEEE Std 1003.1-200x, with the information presented in the
615 appropriate sections and subsections. Sections and subsections that consist solely of subordinate
616 section titles, with no other information, are not required. The conformance document shall not
617 contain information about extended facilities or capabilities outside the scope of
618 IEEE Std 1003.1-200x.

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

624 The conformance document shall describe the limit values found in the headers **<limits.h>** (on
625 page 245) and **<unistd.h>** (on page 398), stating values, the conditions under which those values
626 may change, and the limits of such variations, if any.

627 The conformance document shall describe the behavior of the implementation for all
628 implementation-defined features defined in IEEE Std 1003.1-200x. This requirement shall be met
629 by listing these features and providing either a specific reference to the system documentation or
630 providing full syntax and semantics of these features. When the value or behavior in the

631 implementation is designed to be variable or customized on each instantiation of the system, the
 632 implementation provider shall document the nature and permissible ranges of this variation.

633 The conformance document may specify the behavior of the implementation for those features
 634 where IEEE Std 1003.1-200x states that implementations may vary or where features are
 635 identified as undefined or unspecified.

636 The conformance document shall not contain documentation other than that specified in the
 637 preceding paragraphs except where such documentation is specifically allowed or required by
 638 other provisions of IEEE Std 1003.1-200x.

639 The phrases “shall document” or “shall be documented” in IEEE Std 1003.1-200x mean that
 640 documentation of the feature shall appear in the conformance document, as described
 641 previously, unless there is an explicit reference in the conformance document to show where the
 642 information can be found in the system documentation.

643 The system documentation should also contain the information found in the conformance
 644 document.

645 2.1.3 POSIX Conformance

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

647 2.1.3.1 POSIX System Interfaces

648 • The system shall support all the mandatory functions and headers defined in |
 649 IEEE Std 1003.1-200x, and shall set the symbolic constant `_POSIX_VERSION` to the value |
 650 `200xxxL`. |

651 • Although all implementations conforming to IEEE Std 1003.1-200x support all the features |
 652 described below, there may be system-dependent or file system-dependent configuration |
 653 procedures that can remove or modify any or all of these features. Such configurations |
 654 should not be made if strict compliance is required.

655 The following symbolic constants shall either be undefined or defined with a value other
 656 than `-1`. If a constant is undefined, an application should use the `sysconf()`, `pathconf()`, or
 657 `fpathconf()` functions, or the `getconf` utility, to determine which features are present on the
 658 system at that time or for the particular pathname in question.

659 — `_POSIX_CHOWN_RESTRICTED`

660 The use of `chown()` is restricted to a process with appropriate privileges, and to changing
 661 the group ID of a file only to the effective group ID of the process or to one of its
 662 supplementary group IDs.

663 — `_POSIX_NO_TRUNC`

664 Pathname components longer than `{NAME_MAX}` generate an error.

665 • The following symbolic constants shall be defined as follows: |

666 • `_POSIX_JOB_CONTROL` shall have a value greater than zero. |

667 • `_POSIX_SAVED_IDS` shall have a value greater than zero. |

668 • `_POSIX_VDISABLE` shall have a value other than `-1`. |

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

- 671 • The system may support one or more options (see Section 2.1.6 (on page 26)) denoted by the
- 672 following symbolic constants:
- 673 — _POSIX_ADVISORY_INFO
- 674 — _POSIX_ASYNCHRONOUS_IO
- 675 — _POSIX_BARRIERS
- 676 — _POSIX_CLOCK_SELECTION
- 677 — _POSIX_CPUTIME
- 678 — _POSIX_FSYNC
- 679 — _POSIX_IPV6
- 680 — _POSIX_MAPPED_FILES
- 681 — _POSIX_MEMLOCK
- 682 — _POSIX_MEMLOCK_RANGE
- 683 — _POSIX_MEMORY_PROTECTION
- 684 — _POSIX_MESSAGE_PASSING
- 685 — _POSIX_MONOTONIC_CLOCK
- 686 — _POSIX_PRIORITIZED_IO
- 687 — _POSIX_PRIORITY_SCHEDULING
- 688 — _POSIX_RAW_SOCKETS
- 689 — _POSIX_REALTIME_SIGNALS
- 690 — _POSIX_SEMAPHORES
- 691 — _POSIX_SHARED_MEMORY_OBJECTS
- 692 — _POSIX_SPAWN
- 693 — _POSIX_SPIN_LOCKS
- 694 — _POSIX_SPARADIC_SERVER
- 695 — _POSIX_SYNCHRONIZED_IO
- 696 — _POSIX_THREAD_ATTR_STACKADDR
- 697 — _POSIX_THREAD_CPUTIME
- 698 — _POSIX_THREAD_ATTR_STACKSIZE
- 699 — _POSIX_THREAD_PRIO_INHERIT
- 700 — _POSIX_THREAD_PRIO_PROTECT
- 701 — _POSIX_THREAD_PRIORITY_SCHEDULING
- 702 — _POSIX_THREAD_PROCESS_SHARED
- 703 — _POSIX_THREAD_SAFE_FUNCTIONS
- 704 — _POSIX_THREAD_SPARADIC_SERVER
- 705 — _POSIX_THREADS

706 — _POSIX_TIMEOUTS
 707 — _POSIX_TIMERS
 708 — _POSIX_TRACE
 709 — _POSIX_TRACE_EVENT_FILTER
 710 — _POSIX_TRACE_INHERIT
 711 — _POSIX_TRACE_LOG
 712 — _POSIX_TYPED_MEMORY_OBJECTS

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

716 XSI • The system may support the XSI extensions (see Section 2.1.5.2 (on page 21)) denoted by the
 717 following symbolic constants:

718 — _XOPEN_CRYPT
 719 — _XOPEN_LEGACY
 720 — _XOPEN_REALTIME
 721 — _XOPEN_REALTIME_THREADS
 722 — _XOPEN_UNIX

723 2.1.3.2 POSIX Shell and Utilities

724 • The system shall provide all the mandatory utilities in the Shell and Utilities volume of
 725 IEEE Std 1003.1-200x with all the functional behavior described therein.

726 • The system shall support the Large File capabilities described in the Shell and Utilities
 727 volume of IEEE Std 1003.1-200x.

728 • The system may support one or more options (see Section 2.1.6 (on page 26)) denoted by the
 729 following symbolic constants. (The literal names below apply to the *getconf* utility.)

730 — POSIX2_C_DEV
 731 — POSIX2_CHAR_TERM
 732 — POSIX2_FORT_DEV
 733 — POSIX2_FORT_RUN
 734 — POSIX2_LOCALEDEF
 735 — POSIX2_PBS
 736 — POSIX2_PBS_ACCOUNTING
 737 — POSIX2_PBS_LOCATE
 738 — POSIX2_PBS_MESSAGE
 739 — POSIX2_PBS_TRACK
 740 — POSIX2_SW_DEV
 741 — POSIX2_UPE

- 742 • The system may support the XSI extensions (see Section 2.1.4).

743 Additional language bindings and development utility options may be provided in other related
744 standards or in a future version of IEEE Std 1003.1-200x. In the former case, additional symbolic
745 constants of the same general form as shown in this subsection should be defined by the related
746 standard document and made available to the application without requiring
747 IEEE Std 1003.1-200x to be updated.

748 **2.1.4 XSI Conformance**

749 XSI This section describes the criteria for implementations conforming to the X/Open System
750 Interface extension. This functionality is dependent on the support of the XSI extension (and the
751 rest of this section is not further shaded).

752 IEEE Std 1003.1-200x describes utilities, functions, and facilities offered to application programs
753 by the X/Open System Interface (XSI). An XSI-conforming implementation shall meet the
754 criteria for POSIX conformance and the following requirements.

755 **2.1.4.1 XSI System Interfaces**

- 756 • The system shall support all the functions and headers defined in IEEE Std 1003.1-200x as
757 part of the XSI extension denoted by the symbolic constant `_XOPEN_UNIX` and any
758 extensions marked with the XSI extension marking (see Section 1.5.1 (on page 6)).

- 759 • The system shall support the `mmap()`, `munmap()`, and `msync()` functions.

- 760 • The system shall support the following options defined within IEEE Std 1003.1-200x (see
761 Section 2.1.6 (on page 26)):

762 — `_POSIX_FSYNC`

763 — `_POSIX_MAPPED_FILES`

764 — `_POSIX_MEMORY_PROTECTION`

765 — `_POSIX_THREAD_ATTR_STACKADDR`

766 — `_POSIX_THREAD_ATTR_STACKSIZE`

767 — `_POSIX_THREAD_PROCESS_SHARED`

768 — `_POSIX_THREAD_SAFE_FUNCTIONS`

769 — `_POSIX_THREADS`

- 770 • The system may support the following XSI Option Groups (see Section 2.1.5.2 (on page 21)) |
771 defined within IEEE Std 1003.1-200x:

772 — Encryption

773 — Realtime

774 — Advanced Realtime

775 — Realtime Threads

776 — Advanced Realtime Threads

777 — Tracing

778 — XSI STREAMS

779 — Legacy

780 2.1.4.2 XSI Shell and Utilities Conformance

781 • The system shall support all the utilities defined in the Shell and Utilities volume of
782 IEEE Std 1003.1-200x as part of the XSI extension denoted by the XSI marking in the
783 SYNOPSIS section, and any extensions marked with the XSI extension marking (see Section
784 1.5.1 (on page 6)) within the text.

785 • The system shall support the User Portability Utilities option.

786 • The system shall support creation of locales (see Chapter 7 (on page 119)).

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

788 • The XSI Development Utilities option may be supported. It consists of the following software
789 development utilities:

790	<i>admin</i>	<i>delta</i>	<i>rmdel</i>	<i>val</i>
791	<i>cflow</i>	<i>get</i>	<i>sact</i>	<i>what</i>
792	<i>ctags</i>	<i>m4</i>	<i>sccs</i>	
793	<i>cxref</i>	<i>prs</i>	<i>unget</i>	

794 • Within the utilities that are provided, functionality marked by the code OF (see Section 1.5.1
795 (on page 6)) need not be provided.

796 2.1.5 Option Groups

797 An Option Group is a group of related functions or options defined within the System Interfaces
798 volume of IEEE Std 1003.1-200x.

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

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

803 2.1.5.1 Subprofiling Considerations

804 Profiling standards supporting functional requirements less than that required in
805 IEEE Std 1003.1-200x may subset both mandatory and optional functionality required for POSIX
806 Conformance (see Section 2.1.3 (on page 16)) or XSI Conformance (see Section 2.1.4 (on page
807 19)). Such profiles shall organize the subsets into Subprofiling Option Groups.

808 The Rationale (Informative) volume of IEEE Std 1003.1-200x, Appendix E, Subprofiling
809 Considerations (Informative) describes a representative set of such Subprofiling Option Groups
810 for use by profiles applicable to specialized realtime systems. IEEE Std 1003.1-200x does not
811 require that the presence of Subprofiling Option Groups be testable at compile-time (as symbols
812 defined in any header) or at runtime (via *sysconf()* or *getconf()*).

813 A Subprofiling Option Group may provide basic system functionality that other Subprofiling
814 Option Groups and other options depend upon.³ If a profile of IEEE Std 1003.1-200x does not

815 _____
816 3. As an example, the File System profiling option group provides underlying support for pathname resolution and file creation
817 which are needed by any interface in IEEE Std 1003.1-200x that parses a *path* argument. If a profile requires support for the
818 Device Input and Output profiling option group but does not require support for the File System profiling option group, the
819 profile must specify how pathname resolution is to behave in that profile, how the *O_CREAT* flag to *open()* is to be handled (and
820 the use of the character 'a' in the *mode* argument of *open()* when a filename argument names a file that does not exist), and
821 specify lots of other details.

822 require an implementation to provide a Subprofiling Option Group that provides features |
 823 utilized by a required Subprofiling Option Group (or option),⁴ the profile shall specify⁵ all of the |
 824 following: |

- 825 • Restricted or altered behavior of interfaces defined in IEEE Std 1003.1-200x that may differ on |
 826 an implementation of the profile
- 827 • Additional behaviors that may produce undefined or unspecified results
- 828 • Additional implementation-defined behavior that implementations shall be required to |
 829 document in the profile's conformance document

830 if any of the above is a result of the profile not requiring an interface required by |
 831 IEEE Std 1003.1-200x. |

832 The following additional rules shall apply to all standard profiles of IEEE Std 1003.1-200x: |

- 833 • Any application that conforms to that profile shall also conform to IEEE Std 1003.1-200x (that |
 834 is, a profile cannot require restricted, altered, or extended behaviors).
- 835 • Any implementation that conforms to IEEE Std 1003.1-200x (including all options required |
 836 by the profile) shall also conform to that profile |

837 2.1.5.2 XSI Option Groups

838 XSI This section describes Option Groups to support the definition of XSI conformance within the |
 839 System Interfaces volume of IEEE Std 1003.1-200x. This functionality is dependent on the |
 840 support of the XSI extension (and the rest of this section is not further shaded). |

841 The following Option Groups are defined.

842 **Encryption**

843 The Encryption Option Group is denoted by the symbolic constant `_XOPEN_CRYPT`. It includes |
 844 the following functions:

845 `crypt()`, `encrypt()`, `setkey()`

846 These functions are marked CRYPT.

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

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

853 _____
 854 4. As an example, IEEE Std 1003.1-200x requires that implementations claiming to support the Range Memory Locking option also |
 855 support the Process Memory Locking option. A profile could require that the Range Memory Locking option had to be supplied |
 856 without requiring that the Process Memory Locking option be supplied as long as the profile specifies everything an application |
 857 writer or system implementor would have to know to build an application or implementation conforming to the profile.

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

860 **Realtime**861 The Realtime Option Group is denoted by the symbolic constant `_XOPEN_REALTIME`.862 This Option Group includes a set of realtime functions drawn from options within
863 IEEE Std 1003.1-200x (see Section 2.1.6 (on page 26)).864 Where entire functions are included in the Option Group, the NAME section is marked with
865 REALTIME. Where additional semantics have been added to existing pages, the new material is
866 identified by use of the appropriate margin legend for the underlying option defined within
867 IEEE Std 1003.1-200x.868 An implementation that claims conformance to this Option Group shall set |
869 `_XOPEN_REALTIME` to a value other than `-1`. |870 This Option Group consists of the set of the following options from within IEEE Std 1003.1-200x
871 (see Section 2.1.6 (on page 26)):872 `_POSIX_ASYNCHRONOUS_IO`
873 `_POSIX_FSYNC`
874 `_POSIX_MAPPED_FILES`
875 `_POSIX_MEMLOCK`
876 `_POSIX_MEMLOCK_RANGE`
877 `_POSIX_MEMORY_PROTECTION`
878 `_POSIX_MESSAGE_PASSING`
879 `_POSIX_PRIORITIZED_IO`
880 `_POSIX_PRIORITY_SCHEDULING`
881 `_POSIX_REALTIME_SIGNALS`
882 `_POSIX_SEMAPHORES`
883 `_POSIX_SHARED_MEMORY_OBJECTS`
884 `_POSIX_SYNCHRONIZED_IO`
885 `_POSIX_TIMERS`886 If the symbolic constant `_XOPEN_REALTIME` is defined to have a value other than `-1`, then the
887 following symbolic constants shall be defined by the implementation to have the value `200xxxL`: |888 `_POSIX_ASYNCHRONOUS_IO`
889 `_POSIX_MEMLOCK`
890 `_POSIX_MEMLOCK_RANGE`
891 `_POSIX_MESSAGE_PASSING`
892 `_POSIX_PRIORITY_SCHEDULING`
893 `_POSIX_REALTIME_SIGNALS`
894 `_POSIX_SEMAPHORES`
895 `_POSIX_SHARED_MEMORY_OBJECTS`
896 `_POSIX_SYNCHRONIZED_IO`
897 `_POSIX_TIMERS`898 The functionality associated with `_POSIX_MAPPED_FILES`, `_POSIX_MEMORY_PROTECTION`,
899 and `_POSIX_FSYNC` is always supported on XSI-conformant systems.900 Support of `_POSIX_PRIORITIZED_IO` on XSI-conformant systems is optional. If this
901 functionality is supported, then `_POSIX_PRIORITIZED_IO` shall be set to a value other than `-1`.
902 Otherwise, it shall be undefined.903 If `_POSIX_PRIORITIZED_IO` is supported, then asynchronous I/O operations performed by
904 `aio_read()`, `aio_write()`, and `lio_listio()` shall be submitted at a priority equal to the scheduling
905 priority of the process minus `aiocbp->aio_reqprio`. The implementation shall also document for
906 which files I/O prioritization is supported.

907 **Advanced Realtime**

908 An implementation that claims conformance to this Option Group shall also support the
909 Realtime Option Group.

910 Where entire functions are included in the Option Group, the NAME section is marked with
911 ADVANCED REALTIME. Where additional semantics have been added to existing pages, the
912 new material is identified by use of the appropriate margin legend for the underlying option
913 defined within IEEE Std 1003.1-200x.

914 This Option Group consists of the set of the following options from within IEEE Std 1003.1-200x
915 (see Section 2.1.6 (on page 26)):

```
916        _POSIX_ADVISORY_INFO
917        _POSIX_CLOCK_SELECTION
918        _POSIX_CPUTIME
919        _POSIX_MONOTONIC_CLOCK
920        _POSIX_SPAWN
921        _POSIX_SPORADIC_SERVER
922        _POSIX_TIMEOUTS
923        _POSIX_TYPED_MEMORY_OBJECTS
```

924 If the implementation supports the Advanced Realtime Option Group, then the following
925 symbolic constants shall be defined by the implementation to have the value 200xxxL:

```
926        _POSIX_ADVISORY_INFO
927        _POSIX_CLOCK_SELECTION
928        _POSIX_CPUTIME
929        _POSIX_MONOTONIC_CLOCK
930        _POSIX_SPAWN
931        _POSIX_SPORADIC_SERVER
932        _POSIX_TIMEOUTS
933        _POSIX_TYPED_MEMORY_OBJECTS
```

934 If the symbolic constant `_POSIX_SPORADIC_SERVER` is defined, then the symbolic constant
935 `_POSIX_PRIORITY_SCHEDULING` shall also be defined by the implementation to have the
936 value 200xxxL.

937 If the symbolic constant `_POSIX_CPUTIME` is defined, then the symbolic constant
938 `_POSIX_TIMERS` shall also be defined by the implementation to have the value 200xxxL.

939 If the symbolic constant `_POSIX_MONOTONIC_CLOCK` is defined, then the symbolic constant
940 `_POSIX_TIMERS` shall also be defined by the implementation to have the value 200xxxL.

941 If the symbolic constant `_POSIX_CLOCK_SELECTION` is defined, then the symbolic constant
942 `_POSIX_TIMERS` shall also be defined by the implementation to have the value 200xxxL.

943 **Realtime Threads**

944 The Realtime Threads Option Group is denoted by the symbolic constant
945 `_XOPEN_REALTIME_THREADS`.

946 This Option Group consists of the set of the following options from within IEEE Std 1003.1-200x
947 (see Section 2.1.6 (on page 26)):

```
948        _POSIX_THREAD_PRIO_INHERIT
949        _POSIX_THREAD_PRIO_PROTECT
950        _POSIX_THREAD_PRIORITY_SCHEDULING
```

951 Where applicable, whole pages are marked REALTIME THREADS, together with the
952 appropriate option margin legend for the SYNOPSIS section (see Section 1.5.1 (on page 6)).

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

955 If the symbol `_XOPEN_REALTIME_THREADS` is defined to have a value other than `-1`, then the
956 following options shall also be defined by the implementation to have the value `200xxxL`:

957 `_POSIX_THREAD_PRIO_INHERIT`
958 `_POSIX_THREAD_PRIO_PROTECT`
959 `_POSIX_THREAD_PRIORITY_SCHEDULING`

960 **Advanced Realtime Threads**

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

963 Where entire functions are included in the Option Group, the NAME section is marked with
964 `ADVANCED_REALTIME_THREADS`. Where additional semantics have been added to existing
965 pages, the new material is identified by use of the appropriate margin legend for the underlying
966 option defined within IEEE Std 1003.1-200x.

967 This Option Group consists of the set of the following options from within IEEE Std 1003.1-200x
968 (see Section 2.1.6 (on page 26)):

969 `_POSIX_BARRIERS`
970 `_POSIX_SPIN_LOCKS`
971 `_POSIX_THREAD_CPUTIME`
972 `_POSIX_THREAD_SPORADIC_SERVER`

973 If the symbolic constant `_POSIX_THREAD_SPORADIC_SERVER` is defined to have the value
974 `200xxxL`, then the symbolic constant `_POSIX_THREAD_PRIORITY_SCHEDULING` shall also be
975 defined by the implementation to have the value `200xxxL`.

976 If the symbolic constant `_POSIX_THREAD_CPUTIME` is defined to have the value `200xxxL`,
977 then the symbolic constant `_POSIX_TIMERS` shall also be defined by the implementation to have
978 the value `200xxxL`.

979 If the symbolic constant `_POSIX_BARRIERS` is defined to have the value `200xxxL`, then the
980 symbolic constants `_POSIX_THREADS` and `_POSIX_THREAD_SAFE_FUNCTIONS` shall also
981 be defined by the implementation to have the value `200xxxL`.

982 If the symbolic constant `_POSIX_SPIN_LOCKS` is defined to have the value `200xxxL`, then the
983 symbolic constants `_POSIX_THREADS` and `_POSIX_THREAD_SAFE_FUNCTIONS` shall also
984 be defined by the implementation to have the value `200xxxL`.

985 If the implementation supports the Advanced Realtime Threads Option Group, then the
986 following symbolic constants shall be defined by the implementation to have the value `200xxxL`:

987 `_POSIX_BARRIERS`
988 `_POSIX_SPIN_LOCKS`
989 `_POSIX_THREAD_CPUTIME`
990 `_POSIX_THREAD_SPORADIC_SERVER`

991 **Tracing**

992 This Option Group includes a set of tracing functions drawn from options within
993 IEEE Std 1003.1-200x (see Section 2.1.6 (on page 26)).

994 Where entire functions are included in the Option Group, the NAME section is marked with
995 TRACING. Where additional semantics have been added to existing pages, the new material is
996 identified by use of the appropriate margin legend for the underlying option defined within
997 IEEE Std 1003.1-200x.

998 This Option Group consists of the set of the following options from within IEEE Std 1003.1-200x
999 (see Section 2.1.6 (on page 26)):

1000 `_POSIX_TRACE`
1001 `_POSIX_TRACE_EVENT_FILTER`
1002 `_POSIX_TRACE_LOG`
1003 `_POSIX_TRACE_INHERIT`

1004 If the implementation supports the Tracing Option Group, then the following symbolic
1005 constants shall be defined by the implementation to have the value 200xxxL:

1006 `_POSIX_TRACE`
1007 `_POSIX_TRACE_EVENT_FILTER`
1008 `_POSIX_TRACE_LOG`
1009 `_POSIX_TRACE_INHERIT`

1010 **XSI STREAMS**

1011 The XSI STREAMS Option Group is denoted by the symbolic constant `_XOPEN_STREAMS`.

1012 This Option Group includes functionality related to STREAMS, a uniform mechanism for
1013 implementing networking services and other character-based I/O as described in the System
1014 Interfaces volume of IEEE Std 1003.1-200x, Section 2.6, STREAMS.

1015 It includes the following functions:

1016 `fattach()`, `fdetach()`, `getmsg()`, `getpmsg()`, `ioctl()`, `isastream()`, `putmsg()`, `putpmsg()`

1017 and the `<stropts.h>` header.

1018 Where applicable, whole pages are marked STREAMS, together with the appropriate option
1019 margin legend for the SYNOPSIS section (see Section 1.5.1 (on page 6)). Where additional
1020 semantics have been added to existing pages, the new material is identified by use of the
1021 appropriate margin legend for the underlying option defined within IEEE Std 1003.1-200x.

1022 An implementation that claims conformance to this Open Group shall set `_XOPEN_STREAMS`
1023 to a value other than `-1`.

1024 **Legacy**

1025 The Legacy Option Group is denoted by the symbolic constant `_XOPEN_LEGACY`.

1026 The Legacy Option Group includes the functions and headers which were mandatory in
1027 previous versions of IEEE Std 1003.1-200x but are optional in this version.

1028 These functions and headers are retained in IEEE Std 1003.1-200x because of their widespread
1029 use. Application writers should not rely on the existence of these functions or headers in new
1030 applications, but should follow the migration path detailed in the APPLICATION USAGE
1031 sections of the relevant pages.

1032 Various factors may have contributed to the decision to mark a function or header LEGACY. In
 1033 all cases, the specific reasons for the withdrawal of a function or header are documented on the
 1034 relevant pages.

1035 Once a function or header is marked LEGACY, no modifications are made to the specifications
 1036 of such functions or headers other than to the APPLICATION USAGE sections of the relevant
 1037 pages.

1038 The functions and headers which form this Option Group are as follows:

1039 *bcmp()*, *bcopy()*, *bzero()*, *ecvt()*, *fcvt()*, *ftime()*, *gcvt()*, *getwd()*, *index()*, *mktemp()*, *rindex()*,
 1040 *utimes()*, *wcswcs()*

1041 An implementation that claims conformance to this Option Group shall set `_XOPEN_LEGACY` |
 1042 to a value other than `-1`. |

1043 2.1.6 Options

1044 The symbolic constants defined in `<unistd.h>`, **Constants for Options and Option Groups** (on |
 1045 page 398) reflect implementation options for IEEE Std 1003.1-200x. These symbols can be used |
 1046 by the application to determine which optional facilities are present on the implementation. The |
 1047 *sysconf()* function defined in the System Interfaces volume of IEEE Std 1003.1-200x or the *getconf* |
 1048 utility defined in the Shell and Utilities volume of IEEE Std 1003.1-200x can be used to retrieve |
 1049 the value of each symbol on each specific implementation to determine whether the option is |
 1050 supported. |

1051 Where an option is not supported, the associated utilities, functions, or facilities need not be
 1052 present.

1053 Margin codes are defined for each option (see Section 1.5.1 (on page 6)).

1054 2.1.6.1 System Interfaces

1055 Refer to `<unistd.h>`, **Constants for Options and Option Groups** (on page 398) for the list of |
 1056 options. |

1057 2.1.6.2 Shell and Utilities

1058 Each of these symbols shall be considered valid names by the implementation. Refer to |
 1059 `<unistd.h>`, **Constants for Options and Option Groups** (on page 398). |

1060 The literal names shown below apply only to the *getconf* utility.

1061 CD POSIX2_C_DEV

1062 The system supports the C-Language Development Utilities option.

1063 The utilities in the C-Language Development Utilities option are used for the development
 1064 of C-language applications, including compilation or translation of C source code and
 1065 complex program generators for simple lexical tasks and processing of context-free
 1066 grammars.

1067 The utilities listed below may be provided by a conforming system; however, any system
 1068 claiming conformance to the C-Language Development Utilities option shall provide all of
 1069 the utilities listed.

1070 *c99*
 1071 *lex*
 1072 *yacc*

1073		POSIX2_CHAR_TERM
1074		The system supports the Terminal Characteristics option. This value need not be present on
1075		a system not supporting the User Portability Utilities option.
1076		Where applicable, the dependency is noted within the description of the utility.
1077		This option applies only to systems supporting the User Portability Utilities option. If
1078		supported, then the system supports at least one terminal type capable of all operations
1079		described in IEEE Std 1003.1-200x; see Section 10.2 (on page 181).
1080	FD	POSIX2_FORT_DEV
1081		The system supports the FORTRAN Development Utilities option.
1082		The <i>fort77</i> FORTRAN compiler is the only utility in the FORTRAN Development Utilities
1083		option. This is used for the development of FORTRAN language applications, including
1084		compilation or translation of FORTRAN source code.
1085		The <i>fort77</i> utility may be provided by a conforming system; however, any system claiming
1086		conformance to the FORTRAN Development Utilities option shall provide the <i>fort77</i> utility.
1087	FR	POSIX2_FORT_RUN
1088		The system supports the FORTRAN Runtime Utilities option.
1089		The <i>asa</i> utility is the only utility in the FORTRAN Runtime Utilities option.
1090		The <i>asa</i> utility may be provided by a conforming system; however, any system claiming
1091		conformance to the FORTRAN Runtime Utilities option shall provide the <i>asa</i> utility.
1092		POSIX2_LOCALEDEF
1093		The system supports the Locale Creation Utilities option.
1094		If supported, the system supports the creation of locales as described in the <i>localedef</i> utility.
1095		The <i>localedef</i> utility may be provided by a conforming system; however, any system
1096		claiming conformance to the Locale Creation Utilities option shall provide the <i>localedef</i>
1097		utility.
1098	BE	POSIX2_PBS
1099		The system supports the Batch Environment Services and Utilities option (see the Shell and
1100		Utilities volume of IEEE Std 1003.1-200x, Chapter 3, Batch Environment Services).
1101		Note: The Batch Environment Services and Utilities option is a combination of mandatory and
1102		optional batch services and utilities. The POSIX_PBS symbolic constant implies the
1103		system supports all the mandatory batch services and utilities.
1104		POSIX2_PBS_ACCOUNTING
1105		The system supports the Batch Accounting option.
1106		POSIX2_PBS_CHECKPOINT
1107		The system supports the Batch Checkpoint/Restart option.
1108		POSIX2_PBS_LOCATE
1109		The system supports the Locate Batch Job Request option.
1110		POSIX2_PBS_MESSAGE
1111		The system supports the Batch Job Message Request option.
1112		POSIX2_PBS_TRACK
1113		The system supports the Track Batch Job Request option.
1114	SD	POSIX2_SW_DEV
1115		The system supports the Software Development Utilities option.

1116 The utilities in the Software Development Utilities option are used for the development of
 1117 applications, including compilation or translation of source code, the creation and
 1118 maintenance of library archives, and the maintenance of groups of inter-dependent
 1119 programs.

1120 The utilities listed below may be provided by the conforming system; however, any system
 1121 claiming conformance to the Software Development Utilities option shall provide all of the
 1122 utilities listed here.

1123 *ar*
 1124 *make*
 1125 *nm*
 1126 *strip*

1127 UP POSIX2_UPE

1128 The system supports the User Portability Utilities option.

1129 The utilities in the User Portability Utilities option shall be implemented on all systems that
 1130 claim conformance to this option. Certain utilities are noted as having features that cannot
 1131 be implemented on all terminal types; if the POSIX2_CHAR_TERM option is supported, the
 1132 system shall support all such features on at least one terminal type; see Section 10.2 (on
 1133 page 181).

1134 Some of the utilities are required only on systems that also support the Software
 1135 Development Utilities option, or the character-at-a-time terminal option (see Section 10.2
 1136 (on page 181)); such utilities have this noted in their DESCRIPTION sections. All of the
 1137 other utilities listed are required only on systems that claim conformance to the User
 1138 Portability Utilities option.

1139	<i>alias</i>	<i>expand</i>	<i>nm</i>	<i>unalias</i>
1140	<i>at</i>	<i>fc</i>	<i>patch</i>	<i>unexpand</i>
1141	<i>batch</i>	<i>fg</i>	<i>ps</i>	<i>uudecode</i>
1142	<i>bg</i>	<i>file</i>	<i>renice</i>	<i>uuencode</i>
1143	<i>crontab</i>	<i>jobs</i>	<i>split</i>	<i>vi</i>
1144	<i>split</i>	<i>man</i>	<i>strings</i>	<i>who</i>
1145	<i>ctags</i>	<i>mesg</i>	<i>tabs</i>	<i>write</i>
1146	<i>df</i>	<i>more</i>	<i>talk</i>	
1147	<i>du</i>	<i>newgrp</i>	<i>time</i>	
1148	<i>ex</i>	<i>nice</i>	<i>tput</i>	

1149 2.2 Application Conformance

1150 All applications claiming conformance to IEEE Std 1003.1-200x shall use only language-
 1151 dependent services for the C programming language described in Section 2.3 (on page 31), shall
 1152 use only the utilities and facilities defined in the Shell and Utilities volume of
 1153 IEEE Std 1003.1-200x, and shall fall within one of the following categories.

1154 2.2.1 Strictly Conforming POSIX Application

1155 A Strictly Conforming POSIX Application is an application that requires only the facilities
1156 described in IEEE Std 1003.1-200x. Such an application:

- 1157 1. Shall accept any implementation behavior that results from actions it takes in areas
1158 described in IEEE Std 1003.1-200x as *implementation-defined* or *unspecified*, or where
1159 IEEE Std 1003.1-200x indicates that implementations may vary
- 1160 2. Shall not perform any actions that are described as producing *undefined* results
- 1161 3. For symbolic constants, shall accept any value in the range permitted by
1162 IEEE Std 1003.1-200x, but shall not rely on any value in the range being greater than the
1163 minimums listed or being less than the maximums listed in IEEE Std 1003.1-200x
- 1164 4. Shall not use facilities designated as *obsolescent*
- 1165 5. Is required to tolerate and permitted to adapt to the presence or absence of optional
1166 facilities whose availability is indicated by Section 2.1.3 (on page 16)
- 1167 6. For the C programming language, shall not produce any output dependent on any
1168 behavior described in the ISO/IEC 9899:1999 standard as *unspecified*, *undefined*, or
1169 *implementation-defined*, unless the System Interfaces volume of IEEE Std 1003.1-200x
1170 specifies the behavior
- 1171 7. For the C programming language, shall not exceed any minimum implementation limit
1172 defined in the ISO/IEC 9899:1999 standard, unless the System Interfaces volume of
1173 IEEE Std 1003.1-200x specifies a higher minimum implementation limit
- 1174 8. For the C programming language, shall define `_POSIX_C_SOURCE` to be 200xxxL before
1175 any header is included

1176 Within IEEE Std 1003.1-200x, any restrictions placed upon a Conforming POSIX Application
1177 shall restrict a Strictly Conforming POSIX Application.

1178 2.2.2 Conforming POSIX Application

1179 2.2.2.1 ISO/IEC Conforming POSIX Application

1180 An ISO/IEC Conforming POSIX Application is an application that uses only the facilities
1181 described in IEEE Std 1003.1-200x and approved Conforming Language bindings for any ISO or
1182 IEC standard. Such an application shall include a statement of conformance that documents all
1183 options and limit dependencies, and all other ISO or IEC standards used.

1184 2.2.2.2 <National Body> Conforming POSIX Application

1185 A <National Body> Conforming POSIX Application differs from an ISO/IEC Conforming
1186 POSIX Application in that it also may use specific standards of a single ISO/IEC member body
1187 referred to here as <National Body>. Such an application shall include a statement of
1188 conformance that documents all options and limit dependencies, and all other <National Body>
1189 standards used.

1190 2.2.3 Conforming POSIX Application Using Extensions

1191 A Conforming POSIX Application Using Extensions is an application that differs from a
1192 Conforming POSIX Application only in that it uses non-standard facilities that are consistent
1193 with IEEE Std 1003.1-200x. Such an application shall fully document its requirements for these
1194 extended facilities, in addition to the documentation required of a Conforming POSIX
1195 Application. A Conforming POSIX Application Using Extensions shall be either an ISO/IEC
1196 Conforming POSIX Application Using Extensions or a <National Body> Conforming POSIX
1197 Application Using Extensions (see Section 2.2.2.1 (on page 29) and Section 2.2.2.2 (on page 29)).

1198 2.2.4 Strictly Conforming XSI Application

1199 A Strictly Conforming XSI Application is an application that requires only the facilities described
1200 in IEEE Std 1003.1-200x. Such an application:

- 1201 1. Shall accept any implementation behavior that results from actions it takes in areas
1202 described in IEEE Std 1003.1-200x as *implementation-defined* or *unspecified*, or where
1203 IEEE Std 1003.1-200x indicates that implementations may vary
- 1204 2. Shall not perform any actions that are described as producing *undefined* results
- 1205 3. For symbolic constants, shall accept any value in the range permitted by
1206 IEEE Std 1003.1-200x, but shall not rely on any value in the range being greater than the
1207 minimums listed or being less than the maximums listed in IEEE Std 1003.1-200x
- 1208 4. Shall not use facilities designated as *obsolescent*
- 1209 5. Is required to tolerate and permitted to adapt to the presence or absence of optional
1210 facilities whose availability is indicated by Section 2.1.4 (on page 19)
- 1211 6. For the C programming language, shall not produce any output dependent on any
1212 behavior described in the ISO C standard as *unspecified*, *undefined*, or *implementation-*
1213 *defined*, unless the System Interfaces volume of IEEE Std 1003.1-200x specifies the behavior
- 1214 7. For the C programming language, shall not exceed any minimum implementation limit
1215 defined in the ISO C standard, unless the System Interfaces volume of
1216 IEEE Std 1003.1-200x specifies a higher minimum implementation limit
- 1217 8. For the C programming language, shall define `_XOPEN_SOURCE` to be 600 before any
1218 header is included

1219 Within IEEE Std 1003.1-200x, any restrictions placed upon a Conforming POSIX Application
1220 shall restrict a Strictly Conforming XSI Application.

1221 2.2.5 Conforming XSI Application Using Extensions

1222 A Conforming XSI Application Using Extensions is an application that differs from a Strictly
1223 Conforming XSI Application only in that it uses non-standard facilities that are consistent with
1224 IEEE Std 1003.1-200x. Such an application shall fully document its requirements for these
1225 extended facilities, in addition to the documentation required of a Strictly Conforming XSI
1226 Application.

1227 2.3 Language-Dependent Services for the C Programming Language

1228 Implementors seeking to claim conformance using the ISO C standard shall claim POSIX
1229 conformance as described in Section 2.1.3 (on page 16).

1230 2.4 Other Language-Related Specifications

1231 IEEE Std 1003.1-200x is currently specified in terms of the shell command language and ISO C.
1232 Bindings to other programming languages are being developed.

1233 If conformance to IEEE Std 1003.1-200x is claimed for implementation of any programming
1234 language, the implementation of that language shall support the use of external symbols distinct
1235 to at least 31 bytes in length in the source program text. (That is, identifiers that differ at or
1236 before the thirty-first byte shall be distinct.) If a national or international standard governing a
1237 language defines a maximum length that is less than this value, the language-defined maximum
1238 shall be supported. External symbols that differ only by case shall be distinct when the character
1239 set in use distinguishes uppercase and lowercase characters and the language permits (or
1240 requires) uppercase and lowercase characters to be distinct in external symbols.

Definitions

1242

1243 For the purposes of IEEE Std 1003.1-200x, the terms and definitions given in Chapter 3 apply.

1244 **Note:** No shading to denote extensions or options occurs in this chapter. Where the terms and
 1245 definitions given in this chapter are used elsewhere in text related to extensions and options,
 1246 they are shaded as appropriate.

1247 **3.1 Abortive Release**

1248 An abrupt termination of a network connection that may result in the loss of data.

1249 **3.2 Absolute Pathname**

1250 A pathname beginning with a single or more than two slashes; see also Section 3.266 (on page
 1251 69).

1252 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 98).

1253 **3.3 Access Mode**

1254 A particular form of access permitted to a file. |

1255 **3.4 Additional File Access Control Mechanism** |

1256 An implementation-defined mechanism that is layered upon the access control mechanisms |
 1257 defined here, but which do not grant permissions beyond those defined herein, although they |
 1258 may further restrict them.

1259 **Note:** File Access Permissions are defined in detail in Section 4.4 (on page 95).

1260 **3.5 Address Space**

1261 The memory locations that can be referenced by a process or the threads of a process.

1262 **3.6 Advisory Information**

1263 An interface that advises the implementation on (portable) application behavior so that it can
 1264 optimize the system. |

1265 **3.7 Affirmative Response** |

1266 An input string that matches one of the responses acceptable to the *LC_MESSAGES* category |
 1267 keyword **yesexpr**, matching an extended regular expression in the current locale.

1268 **Note:** The *LC_MESSAGES* category is defined in detail in Section 7.3.6 (on page 148).

1269 **3.8 Alert**

1270 To cause the user's terminal to give some audible or visual indication that an error or some other
1271 event has occurred. When the standard output is directed to a terminal device, the method for
1272 alerting the terminal user is unspecified. When the standard output is not directed to a terminal
1273 device, the alert is accomplished by writing the <alert> to standard output (unless the utility
1274 description indicates that the use of standard output produces undefined results in this case).

1275 **3.9 Alert Character (<alert>)**

1276 A character that in the output stream should cause a terminal to alert its user via a visual or
1277 audible notification. It is the character designated by '\a' in the C language. It is unspecified
1278 whether this character is the exact sequence transmitted to an output device by the system to
1279 accomplish the alert function.

1280 **3.10 Alias Name**

1281 In the shell command language, a word consisting solely of underscores, digits, and alphabetic
1282 characters from the portable character set and any of the following characters: '!', '%', '&', '@', and '^'.

1283 Implementations may allow other characters within alias names as an extension.

1284 **Note:** The portable character set is defined in detail in Section 6.1 (on page 111).

1285 **3.11 Alignment**

1286 A requirement that objects of a particular type be located on storage boundaries with addresses
1287 that are particular multiples of a byte address.

1288 **Note:** See also the ISO C standard, Section B3.

1289 **3.12 Alternate File Access Control Mechanism**

1290 An implementation-defined mechanism that is independent of the access control mechanisms
1291 defined herein, and which if enabled on a file may either restrict or extend the permissions of a
1292 given user. IEEE Std 1003.1-200x defines when such mechanisms can be enabled and when they
1293 are disabled.

1294 **Note:** File Access Permissions are defined in detail in Section 4.4 (on page 95).

1295 **3.13 Alternate Signal Stack**

1296 Memory associated with a thread, established upon request by the implementation for a thread,
1297 separate from the thread signal stack, in which signal handlers responding to signals sent to that
1298 thread may be executed.

1299 3.14 Ancillary Data |

1300 Protocol-specific, local system-specific, or optional information. The information can be both |
1301 local or end-to-end significant, header information, part of a data portion, protocol-specific, and |
1302 implementation or system-specific. |

1303 3.15 Angle Brackets |

1304 The characters '`<`' (left-angle-bracket) and '`>`' (right-angle-bracket). When used in the phrase |
1305 "enclosed in angle brackets", the symbol '`<`' immediately precedes the object to be enclosed, |
1306 and '`>`' immediately follows it. When describing these characters in the portable character set, |
1307 the names `<less-than-sign>` and `<greater-than-sign>` are used. |

1308 3.16 Application

1309 A computer program that performs some desired function.

1310 3.17 Application Address

1311 Endpoint address of a specific application.

1312 3.18 Application Program Interface (API)

1313 The definition of syntax and semantics for providing computer system services. |

1314 3.19 Appropriate Privileges |

1315 An implementation-defined means of associating privileges with a process with regard to the |
1316 function calls, function call options, and the commands that need special privileges. There may |
1317 be zero or more such means. These means (or lack thereof) are described in the conformance |
1318 document. |

1319 **Note:** Function calls are defined in the System Interfaces volume of IEEE Std 1003.1-200x, and |
1320 commands are defined in the Shell and Utilities volume of IEEE Std 1003.1-200x. |

1321 3.20 Argument |

1322 In the shell command language, a parameter passed to a utility as the equivalent of a single |
1323 string in the `argv` array created by one of the `exec` functions. An argument is one of the options, |
1324 option-arguments, or operands following the command name. |

1325 **Note:** The Utility Argument Syntax is defined in detail in Section 12.1 (on page 197) and the Shell and |
1326 Utilities volume of IEEE Std 1003.1-200x, Section 2.9.1.1, Command Search and Execution. |

1327 In the C language, an expression in a function call expression or a sequence of preprocessing |
1328 tokens in a function-like macro invocation. |

1329 3.21 Arm (a Timer)

1330 To start a timer measuring the passage of time, enabling notifying a process when the specified
1331 time or time interval has passed.

1332 3.22 Asterisk

1333 The character ' * '.

1334 3.23 Async-Cancel-Safe Function

1335 A function that may be safely invoked by an application while the asynchronous form of
1336 cancelation is enabled. No function is async-cancel-safe unless explicitly described as such.

1337 3.24 Asynchronous Events

1338 Events that occur independently of the execution of the application.

1339 3.25 Asynchronous Input and Output

1340 A functionality enhancement to allow an application process to queue data input and output
1341 commands with asynchronous notification of completion. |

1342 3.26 Async-Signal-Safe Function

1343 A function that may be invoked, without restriction, from signal-catching functions. No function
1344 is async-signal-safe unless explicitly described as such. |

1345 3.27 Asynchronously-Generated Signal |

1346 A signal that is not attributable to a specific thread. Examples are signals sent via *kill()*, signals |
1347 sent from the keyboard, and signals delivered to process groups. Being asynchronous is a
1348 property of how the signal was generated and not a property of the signal number. All signals
1349 may be generated asynchronously.

1350 **Note:** The *kill()* function is defined in detail in the System Interfaces volume of IEEE Std 1003.1-200x.

1351 3.28 Asynchronous I/O Operation |

1352 An I/O operation that does not of itself cause the thread requesting the I/O to be blocked from |
1353 further use of the processor.

1354 This implies that the process and the I/O operation may be running concurrently.

1355 **3.29 Asynchronous I/O Completion**

1356 For an asynchronous read or write operation, when a corresponding synchronous read or write
1357 would have completed and when any associated status fields have been updated.

1358 **3.30 Authentication**

1359 The process of validating a user or process to verify that the user or process is not a counterfeit. |

1360 **3.31 Authorization** |

1361 The process of verifying that a user or process has permission to use a resource in the manner |
1362 requested.

1363 To ensure security, the user or process would also need to be authenticated before granting
1364 access.

1365 **3.32 Background Job**

1366 See *Background Process Group* in Section 3.34.

1367 **3.33 Background Process**

1368 A process that is a member of a background process group.

1369 **3.34 Background Process Group (or Background Job)**

1370 Any process group, other than a foreground process group, that is a member of a session that
1371 has established a connection with a controlling terminal.

1372 **3.35 Backquote**

1373 The character ' ` ', also known as a *grave accent*.

1374 **3.36 Backslash**

1375 The character ' \ ', also known as a *reverse solidus*. |

1376 **3.37 Backspace Character (<backspace>)** |

1377 A character that, in the output stream, should cause printing (or displaying) to occur one column |
1378 position previous to the position about to be printed. If the position about to be printed is at the
1379 beginning of the current line, the behavior is unspecified. It is the character designated by ' \b ' |
1380 in the C language. It is unspecified whether this character is the exact sequence transmitted to an
1381 output device by the system to accomplish the backspace function. The <backspace> defined

1382 here is not necessarily the ERASE special character.

1383 **Note:** Special Characters are defined in detail in Section 11.1.9 (on page 187).

1384 **3.38 Barrier**

1385 A synchronization object that allows multiple threads to synchronize at a particular point in
1386 their execution.

1387 **3.39 Base Character**

1388 One of the set of characters defined in the Latin alphabet. In Western European languages other
1389 than English, these characters are commonly used with diacritical marks (accents, cedilla, and so
1390 on) to extend the range of characters in an alphabet.

1391 **3.40 Basename**

1392 The final, or only, filename in a pathname. |

1393 **3.41 Basic Regular Expression (BRE)** |

1394 A regular expression (see Section 3.316 (on page 76)) used by the majority of utilities that select |
1395 strings from a set of character strings. |

1396 **Note:** Basic Regular Expressions are described in detail in Section 9.3 (on page 167). |

1397 **3.42 Batch Access List** |

1398 A list of user IDs and group IDs of those users and groups authorized to place batch jobs in a |
1399 batch queue. |

1400 A batch access list is associated with a batch queue. A batch server uses the batch access list of a
1401 batch queue as one of the criteria in deciding to put a batch job in a batch queue.

1402 **3.43 Batch Administrator**

1403 A user that is authorized to modify all the attributes of queues and jobs and to change the status |
1404 of a batch server. |

1405 **3.44 Batch Client** |

1406 A computational entity that utilizes batch services by making requests of batch servers. |

1407 Batch clients often provide the means by which users access batch services, although a batch
1408 server may act as a batch client by virtue of making requests of another batch server. |

1409 3.45 Batch Destination |

1410 The batch server in a batch system to which a batch job should be sent for processing. |

1411 Acceptance of a batch job at a batch destination is the responsibility of a receiving batch server.
1412 A batch destination may consist of a batch server-specific portion, a network-wide portion, or
1413 both. The batch server-specific portion is referred to as the *batch queue*. The network-wide
1414 portion is referred to as a *batch server name*. |

1415 3.46 Batch Destination Identifier |

1416 A string that identifies a specific batch destination. |

1417 A string of characters in the portable character set used to specify a particular batch destination. |

1418 **Note:** The portable character set is defined in detail in Section 6.1 (on page 111). |

1419 3.47 Batch Directive |

1420 A line from a file that is interpreted by the batch server. The line is usually in the form of a
1421 comment and is an additional means of passing options to the *qsub* utility. |

1422 **Note:** The *qsub* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x. |

1423 3.48 Batch Job |

1424 A set of computational tasks for a computing system. |

1425 Batch jobs are managed by batch servers. |

1426 Once created, a batch job may be executing or pending execution. A batch job that is executing
1427 has an associated session leader (a process) that initiates and monitors the computational tasks
1428 of the batch job. |

1429 3.49 Batch Job Attribute |

1430 A named data type whose value affects the processing of a batch job. |

1431 The values of the attributes of a batch job affect the processing of that job by the batch server
1432 that manages the batch job. |

1433 3.50 Batch Job Identifier |

1434 A unique name for a batch job. A name that is unique among all other batch job identifiers in a
1435 batch system and that identifies the batch server to which the batch job was originally
1436 submitted. |

1437 3.51 Batch Job Name |

1438 A label that is an attribute of a batch job. The batch job name is not necessarily unique. |

1439 3.52 Batch Job Owner |

1440 The *username@hostname* of the user submitting the batch job, where *username* is a user name (see |
1441 also Section 3.426 (on page 91)) and *hostname* is a network host name. |

1442 3.53 Batch Job Priority |

1443 A value specified by the user that may be used by an implementation to determine the order in |
1444 which batch jobs are selected to be executed. Job priority has a numeric value in the range -1 024 |
1445 to 1 023. |

1446 **Note:** The batch job priority is not the execution priority (nice value) of the batch job. |

1447 3.54 Batch Job State |

1448 An attribute of a batch job which determines the types of requests that the batch server that |
1449 manages the batch job can accept for the batch job. Valid states include QUEUED, RUNNING, |
1450 HELD, WAITING, EXITING, and TRANSITING. |

1451 3.55 Batch Name Service

1452 A service that assigns batch names that are unique within the batch name space, and that can |
1453 translate a unique batch name into the location of the named batch entity. |

1454 3.56 Batch Name Space

1455 The environment within which a batch name is known to be unique. |

1456 3.57 Batch Node |

1457 A host containing part or all of a batch system. |

1458 A batch node is a host meeting at least one of the following conditions:

- 1459 • Capable of executing a batch client
- 1460 • Contains a routing batch queue
- 1461 • Contains an execution batch queue

1462 3.58 Batch Operator

1463 A user that is authorized to modify some, but not all, of the attributes of jobs and queues, and |
1464 may change the status of the batch server. |

1465 3.59 Batch Queue |

1466 A manageable object that represents a set of batch jobs and is managed by a single batch server. |

1467 **Note:** A set of batch jobs is called a batch queue largely for historical reasons. Jobs are selected from
1468 the batch queue for execution based on attributes such as priority, resource requirements, and
1469 hold conditions.
1470 See also the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 3.1.2, Batch Queues.

1471 **3.60 Batch Queue Attribute**

1472 A named data type whose value affects the processing of all batch jobs that are members of the
1473 batch queue.

1474 A batch queue has attributes that affect the processing of batch jobs that are members of the
1475 batch queue.

1476 **3.61 Batch Queue Position**

1477 The place, relative to other jobs in the batch queue, occupied by a particular job in a batch queue.
1478 This is defined in part by submission time and priority; see also Section 3.62.

1479 **3.62 Batch Queue Priority**

1480 The maximum job priority allowed for any batch job in a given batch queue.

1481 The batch queue priority is set and may be changed by users with appropriate privilege. The
1482 priority is bounded in an implementation-defined manner.

1483 **3.63 Batch Rerunability**

1484 An attribute of a batch job indicating that it may be rerun after an abnormal termination from
1485 the beginning without affecting the validity of the results.

1486 **3.64 Batch Restart**

1487 The action of resuming the processing of a batch job from the point of the last checkpoint.
1488 Typically, this is done if the batch job has been interrupted because of a system failure.

1489 **3.65 Batch Server**

1490 A computational entity that provides batch services.

1491 **3.66 Batch Server Name**

1492 A string of characters in the portable character set used to specify a particular server in a
1493 network.

1494 **Note:** The portable character set is defined in detail in Section 6.1 (on page 111).

1495 3.67 Batch Service |

1496 Computational and organizational services performed by a batch system on behalf of batch jobs. |

1497 Batch services are of two types: *requested* and *deferred*.

1498 **Note:** Batch Services are listed in the Shell and Utilities volume of IEEE Std 1003.1-200x, Table 3-5,
1499 Batch Services Summary.

1500 3.68 Batch Service Request |

1501 A solicitation of services from a batch client to a batch server. |

1502 A batch service request may entail the exchange of any number of messages between the batch
1503 client and the batch server.

1504 When naming specific types of service requests, the term request is qualified by the type of
1505 request, as in *Queue Batch Job Request* and *Delete Batch Job Request*.

1506 3.69 Batch Submission

1507 The process by which a batch client requests that a batch server create a batch job via a *Queue Job*
1508 *Request* to perform a specified computational task.

1509 3.70 Batch System

1510 A collection of one or more batch servers.

1511 3.71 Batch Target User

1512 The name of a user on the batch destination batch server.

1513 The target user is the user name under whose account the batch job is to execute on the
1514 destination batch server.

1515 3.72 Batch User

1516 A user who is authorized to make use of batch services. |

1517 3.73 Bind

1518 The process of assigning a network address to an endpoint. |

1519 3.74 Blank Character (<blank>) |

1520 One of the characters that belong to the **blank** character class as defined via the *LC_CTYPE*
1521 category in the current locale. In the POSIX locale, a <blank> is either a <tab> or a <space>.

1522 3.75 Blank Line

1523 A line consisting solely of zero or more <blank>s terminated by a <newline>; see also Section
1524 3.144 (on page 52).

1525 3.76 Blocked Process (or Thread)

1526 A process (or thread) that is waiting for some condition (other than the availability of a
1527 processor) to be satisfied before it can continue execution.

1528 3.77 Blocking

1529 A property of an open file description that causes function calls associated with it to wait for the
1530 requested action to be performed before returning.

1531 3.78 Block-Mode Terminal

1532 A terminal device operating in a mode incapable of the character-at-a-time input and output
1533 operations described by some of the standard utilities.

1534 **Note:** Output Devices and Terminal Types are defined in detail in Section 10.2 (on page 181).

1535 3.79 Block Special File

1536 A file that refers to a device. A block special file is normally distinguished from a character
1537 special file by providing access to the device in a manner such that the hardware characteristics
1538 of the device are not visible.

1539 3.80 Braces

1540 The characters ‘{’ (left brace) and ‘}’ (right brace), also known as *curly braces*. When used in
1541 the phrase “enclosed in (curly) braces” the symbol ‘{’ immediately precedes the object to be
1542 enclosed, and ‘}’ immediately follows it. When describing these characters in the portable
1543 character set, the names <left-brace> and <right-brace> are used.

1544 3.81 Brackets

1545 The characters ‘[’ (left-bracket) and ‘]’ (right-bracket), also known as *square brackets*. When
1546 used in the phrase “enclosed in (square) brackets” the symbol ‘[’ immediately precedes the
1547 object to be enclosed, and ‘]’ immediately follows it. When describing these characters in the
1548 portable character set, the names <left-square-bracket> and <right-square-bracket> are used.

1549 3.82 Broadcast

1550 The transfer of data from one endpoint to several endpoints, as described in RFC 919 and
1551 RFC 922.

1552 3.83 Built-In Utility (or Built-In)

1553 A utility implemented within a shell. The utilities referred to as *special built-ins* have special
1554 qualities. Unless qualified, the term built-in includes the special built-in utilities. *Regular built-ins*
1555 are not required to be actually built into the shell on the implementation, but they do have
1556 special command-search qualities.

1557 **Note:** Special Built-In Utilities are defined in detail in the Shell and Utilities volume of
1558 IEEE Std 1003.1-200x, Section 2.14, Special Built-In Utilities.

1559 Regular Built-In Utilities are defined in detail in the Shell and Utilities volume of
1560 IEEE Std 1003.1-200x, Section 2.9.1.1, Command Search and Execution.

1561 3.84 Byte

1562 An individually addressable unit of data storage that is exactly an octet, used to store a character
1563 or a portion of a character; see also Section 3.87. A byte is composed of a contiguous sequence of
1564 8 bits. The least significant bit is called the *low-order* bit; the most significant is called the *high-*
1565 *order* bit.

1566 **Note:** The definition of byte from the ISO C standard is broader than the above and might
1567 accommodate hardware architectures with different sized addressable units than octets.

1568 3.85 Byte Input/Output Functions

1569 The functions that perform byte-oriented input from streams or byte-oriented output to streams:
1570 *fgetc()*, *fgets()*, *fprintf()*, *fputc()*, *fputs()*, *fread()*, *fscanf()*, *fwrite()*, *getc()*, *getchar()*, *gets()*, *printf()*,
1571 *putc()*, *putchar()*, *puts()*, *scanf()*, *ungetc()*, *vfprintf()*, and *vprintf()*.

1572 **Note:** Functions are defined in detail in the System Interfaces volume of IEEE Std 1003.1-200x.

1573 3.86 Carriage-Return Character (<carriage-return>)

1574 A character that in the output stream indicates that printing should start at the beginning of the
1575 same physical line in which the <carriage-return> occurred. It is the character designated by
1576 '\r' in the C language. It is unspecified whether this character is the exact sequence
1577 transmitted to an output device by the system to accomplish the movement to the beginning of
1578 the line.

1579 3.87 Character

1580 A sequence of one or more bytes representing a single graphic symbol or control code.

1581 **Note:** This term corresponds to the ISO C standard term multi-byte character, where a single-byte
1582 character is a special case of a multi-byte character. Unlike the usage in the ISO C standard,
1583 *character* here has no necessary relationship with storage space, and *byte* is used when storage
1584 space is discussed.

1585 See the definition of the portable character set in Section 6.1 (on page 111) for a further
1586 explanation of the graphical representations of (abstract) characters, as opposed to character
1587 encodings.

1588 3.88 Character Array

1589 An array of elements of type **char**.

1590 3.89 Character Class

1591 A named set of characters sharing an attribute associated with the name of the class. The classes
1592 and the characters that they contain are dependent on the value of the *LC_CTYPE* category in the
1593 current locale.

1594 **Note:** The *LC_CTYPE* category is defined in detail in Section 7.3.1 (on page 122).

1595 3.90 Character Set

1596 A finite set of different characters used for the representation, organization, or control of data.

1597 3.91 Character Special File

1598 A file that refers to a device. One specific type of character special file is a terminal device file.

1599 **Note:** The General Terminal Interface is defined in detail in Chapter 11 (on page 183).

1600 3.92 Character String

1601 A contiguous sequence of characters terminated by and including the first null byte.

1602 3.93 Child Process

1603 A new process created (by *fork()* or *spawn()*) by a given process. A child process remains the
1604 child of the creating process as long as both processes continue to exist.

1605 **Note:** The *fork()* and *spawn()* functions are defined in detail in the System Interfaces volume of
1606 IEEE Std 1003.1-200x.

1607 3.94 Circumflex

1608 The character '^'.

1609 3.95 Clock

1610 A software or hardware object that can be used to measure the apparent or actual passage of
1611 time.

1612 The current value of the time measured by a clock can be queried and, possibly, set to a value
1613 within the legal range of the clock.

1614 3.96 Clock Jump

1615 The difference between two successive distinct values of a clock, as observed from the
1616 application via one of the “get time” operations.

1617 3.97 Clock Tick

1618 An interval of time; an implementation-defined number of these occur each second. Clock ticks
1619 are one of the units that may be used to express a value found in type `clock_t`.

1620 3.98 Coded Character Set

1621 A set of unambiguous rules that establishes a character set and the one-to-one relationship
1622 between each character of the set and its bit representation.

1623 3.99 Codeset

1624 The result of applying rules that map a numeric code value to each element of a character set. An
1625 element of a character set may be related to more than one numeric code value but the reverse is
1626 not true. However, for state-dependent encodings the relationship between numeric code values
1627 to elements of a character set may be further controlled by state information. The character set
1628 may contain fewer elements than the total number of possible numeric code values; that is, some
1629 code values may be unassigned.

1630 **Note:** Character Encoding is defined in detail in Section 6.2 (on page 114).

1631 3.100 Collating Element

1632 The smallest entity used to determine the logical ordering of character or wide-character strings;
1633 see also Section 3.102. A collating element consists of either a single character, or two or more
1634 characters collating as a single entity. The value of the `LC_COLLATE` category in the current
1635 locale determines the current set of collating elements.

1636 3.101 Collation

1637 The logical ordering of character or wide-character strings according to defined precedence
1638 rules. These rules identify a collation sequence between the collating elements, and such
1639 additional rules that can be used to order strings consisting of multiple collating elements.

1640 3.102 Collation Sequence

1641 The relative order of collating elements as determined by the setting of the `LC_COLLATE`
1642 category in the current locale. The collation sequence is used for sorting and is determined from
1643 the collating weights assigned to each collating element. In the absence of weights, the collation
1644 sequence is the order in which collating elements are specified between `order_start` and
1645 `order_end` keywords in the `LC_COLLATE` category.

1646 Multi-level sorting is accomplished by assigning elements one or more collation weights, up to
1647 the limit {COLL_WEIGHTS_MAX}. On each level, elements may be given the same weight (at
1648 the primary level, called an equivalence class; see also Section 3.150 (on page 53)) or be omitted
1649 from the sequence. Strings that collate equally using the first assigned weight (primary ordering)
1650 are then compared using the next assigned weight (secondary ordering), and so on.

1651 **Note:** {COLL_WEIGHTS_MAX} is defined in detail in <limits.h>.

1652 **3.103 Column Position**

1653 A unit of horizontal measure related to characters in a line.

1654 It is assumed that each character in a character set has an intrinsic column width independent of
1655 any output device. Each printable character in the portable character set has a column width of
1656 one. The standard utilities, when used as described in IEEE Std 1003.1-200x, assume that all
1657 characters have integral column widths. The column width of a character is not necessarily
1658 related to the internal representation of the character (numbers of bits or bytes).

1659 The column position of a character in a line is defined as one plus the sum of the column widths
1660 of the preceding characters in the line. Column positions are numbered starting from 1.

1661 **3.104 Command**

1662 A directive to the shell to perform a particular task.

1663 **Note:** Shell Commands are defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x,
1664 Section 2.9, Shell Commands.

1665 **3.105 Command Language Interpreter**

1666 An interface that interprets sequences of text input as commands. It may operate on an input
1667 stream or it may interactively prompt and read commands from a terminal. It is possible for
1668 applications to invoke utilities through a number of interfaces, which are collectively considered
1669 to act as command interpreters. The most obvious of these are the *sh* utility and the *system()*
1670 function, although *popen()* and the various forms of *exec* may also be considered to behave as
1671 interpreters.

1672 **Note:** The *sh* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x.

1673 The *system()*, *popen()*, and *exec* functions are defined in detail in the System Interfaces volume
1674 of IEEE Std 1003.1-200x.

1675 **3.106 Composite Graphic Symbol**

1676 A graphic symbol consisting of a combination of two or more other graphic symbols in a single
1677 character position, such as a diacritical mark and a base character.

1678 **3.107 Condition Variable**

1679 A synchronization object which allows a thread to suspend execution, repeatedly, until some
1680 associated predicate becomes true. A thread whose execution is suspended on a condition

1681 variable is said to be blocked on the condition variable.

1682 **3.108 Connection**

1683 An association established between two or more endpoints for the transfer of data

1684 **3.109 Connection Mode**

1685 The transfer of data in the context of a connection; see also Section 3.110.

1686 **3.110 Connectionless Mode**

1687 The transfer of data other than in the context of a connection; see also Section 3.109 and Section
1688 3.123 (on page 49).

1689 **3.111 Control Character**

1690 A character, other than a graphic character, that affects the recording, processing, transmission,
1691 or interpretation of text. |

1692 **3.112 Control Operator**

1693 In the shell command language, a token that performs a control function. It is one of the |
1694 following symbols: |

1695 & && () ; ;; newline | ||

1696 The end-of-input indicator used internally by the shell is also considered a control operator.

1697 **Note:** Token Recognition is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x,
1698 Section 2.3, Token Recognition.

1699 **3.113 Controlling Process**

1700 The session leader that established the connection to the controlling terminal. If the terminal
1701 subsequently ceases to be a controlling terminal for this session, the session leader ceases to be
1702 the controlling process. |

1703 **3.114 Controlling Terminal**

1704 A terminal that is associated with a session. Each session may have at most one controlling |
1705 terminal associated with it, and a controlling terminal is associated with exactly one session. |
1706 Certain input sequences from the controlling terminal cause signals to be sent to all processes in
1707 the process group associated with the controlling terminal.

1708 **Note:** The General Terminal Interface is defined in detail in Chapter 11 (on page 183).

1709 3.115 Conversion Descriptor

1710 A per-process unique value used to identify an open codeset conversion.

1711 3.116 Core File

1712 A file of unspecified format that may be generated when a process terminates abnormally. |

1713 3.117 CPU Time (Execution Time) |

1714 The time spent executing a process or thread, including the time spent executing system services |
1715 on behalf of that process or thread. If the Threads option is supported, then the value of the |
1716 CPU-time clock for a process is implementation-defined. With this definition the sum of all the |
1717 execution times of all the threads in a process might not equal the process execution time, even |
1718 in a single-threaded process, because implementations may differ in how they account for time |
1719 during context switches or for other reasons.

1720 3.118 CPU-Time Clock

1721 A clock that measures the execution time of a particular process or thread.

1722 3.119 CPU-Time Timer

1723 A timer attached to a CPU-time clock.

1724 3.120 Current Job

1725 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities. There
1726 is at most one current job; see also Section 3.203 (on page 60).

1727 3.121 Current Working Directory

1728 See *Working Directory* in Section 3.436 (on page 92).

1729 3.122 Cursor Position

1730 The line and column position on the screen denoted by the terminal's cursor.

1731 3.123 Datagram

1732 A unit of data transferred from one endpoint to another in connectionless mode service.

1733 3.124 Data Segment

1734 Memory associated with a process, that can contain dynamically allocated data. |

1735 3.125 Deferred Batch Service |

1736 A service that is performed as a result of events that are asynchronous with respect to requests. |

1737 **Note:** Once a batch job has been created, it is subject to deferred services.

1738 3.126 Device

1739 A computer peripheral or an object that appears to the application as such.

1740 3.127 Device ID

1741 A non-negative integer used to identify a device.

1742 3.128 Directory

1743 A file that contains directory entries. No two directory entries in the same directory have the
1744 same name.

1745 3.129 Directory Entry (or Link)

1746 An object that associates a filename with a file. Several directory entries can associate names
1747 with the same file.

1748 3.130 Directory Stream

1749 A sequence of all the directory entries in a particular directory. An open directory stream may be
1750 implemented using a file descriptor.

1751 3.131 Disarm (a Timer)

1752 To stop a timer from measuring the passage of time, disabling any future process notifications
1753 (until the timer is armed again).

1754 3.132 Display

1755 To output to the user's terminal. If the output is not directed to a terminal, the results are
1756 undefined.

1757 **3.133 Display Line**

1758 A line of text on a physical device or an emulation thereof. Such a line will have a maximum
1759 number of characters which can be presented.

1760 **Note:** This may also be written as “line on the display”.

1761 **3.134 Dollar Sign**

1762 The character ‘\$’.

1763 **3.135 Dot**

1764 In the context of naming files, the filename consisting of a single dot character (‘.’).

1765 **Note:** In the context of shell special built-in utilities, see *dot* in the Shell and Utilities volume of
1766 IEEE Std 1003.1-200x, Section 2.14, Special Built-In Utilities.

1767 Pathname Resolution is defined in detail in Section 4.11 (on page 98).

1768 **3.136 Dot-Dot**

1769 The filename consisting solely of two dot characters (“..”).

1770 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 98).

1771 **3.137 Double-Quote**

1772 The character ‘”’, also known as *quotation-mark*.

1773 **Note:** The *double* adjective in this term refers to the two strokes in the character glyph.
1774 IEEE Std 1003.1-200x never uses the term double-quote to refer to two apostrophes or
1775 quotation marks.

1776 **3.138 Downshifting**

1777 The conversion of an uppercase character that has a single-character lowercase representation
1778 into this lowercase representation.

1779 **3.139 Driver**

1780 A module that controls data transferred to and received from devices.

1781 **Note:** Drivers are traditionally written to be a part of the system implementation, although they are
1782 frequently written separately from the writing of the implementation. A driver may contain
1783 processor-specific code, and therefore be non-portable.

1784 3.140 Effective Group ID

1785 An attribute of a process that is used in determining various permissions, including file access
1786 permissions; see also Section 3.188 (on page 58).

1787 3.141 Effective User ID

1788 An attribute of a process that is used in determining various permissions, including file access
1789 permissions; see also Section 3.425 (on page 91).

1790 3.142 Eight-Bit Transparency

1791 The ability of a software component to process 8-bit characters without modifying or utilizing
1792 any part of the character in a way that is inconsistent with the rules of the current coded
1793 character set.

1794 3.143 Empty Directory

1795 A directory that contains, at most, directory entries for dot and dot-dot, and has exactly one link |
1796 to it, in dot-dot. No other links to the directory may exist. It is unspecified whether an |
1797 implementation can ever consider the root directory to be empty.

1798 3.144 Empty Line

1799 A line consisting of only a <newline>; see also Section 3.75 (on page 43).

1800 3.145 Empty String (or Null String)

1801 A string whose first byte is a null byte.

1802 3.146 Empty Wide-Character String

1803 A wide-character string whose first element is a null wide-character code. |

1804 3.147 Encoding Rule |

1805 The rules used to convert between wide-character codes and multi-byte character codes. |

1806 **Note:** Stream Orientation and Encoding Rules are defined in detail in the System Interfaces volume
1807 of IEEE Std 1003.1-200x, Section 2.5.2, Stream Orientation and Encoding Rules.

1808 3.148 Entire Regular Expression |

1809 The concatenated set of one or more basic regular expressions or extended regular expressions |
1810 that make up the pattern specified for string selection.

1811 **Note:** Regular Expressions are defined in detail in Chapter 9 (on page 165).

1812 **3.149 Epoch**

1813 The time zero hours, zero minutes, zero seconds, on January 1, 1970 Coordinated Universal Time
1814 (UTC).

1815 **Note:** See also Seconds Since the Epoch defined in Section 4.14 (on page 100).

1816 **3.150 Equivalence Class**

1817 A set of collating elements with the same primary collation weight.

1818 Elements in an equivalence class are typically elements that naturally group together, such as all
1819 accented letters based on the same base letter.

1820 The collation order of elements within an equivalence class is determined by the weights
1821 assigned on any subsequent levels after the primary weight.

1822 **3.151 Era**

1823 A locale-specific method for counting and displaying years.

1824 **Note:** The *LC_TIME* category is defined in detail in Section 7.3.5 (on page 142).

1825 **3.152 Event Management**

1826 The mechanism that enables applications to register for and be made aware of external events
1827 such as data becoming available for reading.

1828 **3.153 Executable File**

1829 A regular file acceptable as a new process image file by the equivalent of the *exec* family of
1830 functions, and thus usable as one form of a utility. The standard utilities described as compilers
1831 can produce executable files, but other unspecified methods of producing executable files may
1832 also be provided. The internal format of an executable file is unspecified, but a conforming
1833 application cannot assume an executable file is a text file.

1834 **3.154 Execute**

1835 To perform command search and execution actions, as defined in the Shell and Utilities volume
1836 of IEEE Std 1003.1-200x; see also Section 3.200 (on page 60).

1837 **Note:** Command Search and Execution is defined in detail in the Shell and Utilities volume of
1838 IEEE Std 1003.1-200x, Section 2.9.1.1, Command Search and Execution.

1839 **3.155 Execution Time**1840 See *CPU Time* in Section 3.117 (on page 49).1841 **3.156 Execution Time Monitoring**1842 A set of execution time monitoring primitives that allow online measuring of thread and process
1843 execution times.1844 **3.157 Expand**

1845 In the shell command language, when not qualified, the act of applying word expansions.

1846 **Note:** Word Expansions are defined in detail in the Shell and Utilities volume of
1847 IEEE Std 1003.1-200x, Section 2.6, Word Expansions.1848 **3.158 Extended Regular Expression (ERE)**1849 A regular expression (see also Section 3.316 (on page 76)) that is an alternative to the Basic
1850 Regular Expression using a more extensive syntax, occasionally used by some utilities.1851 **Note:** Extended Regular Expressions are described in detail in Section 9.4 (on page 171).1852 **3.159 Extended Security Controls**1853 Implementation-defined security controls allowed by the file access permission and appropriate
1854 privilege (see also Section 3.19 (on page 35)) mechanisms, through which an implementation can
1855 support different security policies from those described in IEEE Std 1003.1-200x.1856 **Note:** See also Extended Security Controls defined in Section 4.3 (on page 95).

1857 File Access Permissions are defined in detail in Section 4.4 (on page 95).

1858 **3.160 Feature Test Macro**

1859 A macro used to determine whether a particular set of features is included from a header.

1860 **Note:** See also the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.2, The Compilation
1861 Environment.1862 **3.161 Field**1863 In the shell command language, a unit of text that is the result of parameter expansion,
1864 arithmetic expansion, command substitution, or field splitting. During command processing, the
1865 resulting fields are used as the command name and its arguments.1866 **Note:** Parameter Expansion is defined in detail in the Shell and Utilities volume of
1867 IEEE Std 1003.1-200x, Section 2.6.2, Parameter Expansion.1868 Arithmetic Expansion is defined in detail in the Shell and Utilities volume of
1869 IEEE Std 1003.1-200x, Section 2.6.4, Arithmetic Expansion.

1870 Command Substitution is defined in detail in the Shell and Utilities volume of
1871 IEEE Std 1003.1-200x, Section 2.6.3, Command Substitution.

1872 Field Splitting is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x,
1873 Section 2.6.5, Field Splitting.

1874 For further information on command processing, see the Shell and Utilities volume of
1875 IEEE Std 1003.1-200x, Section 2.9.1, Simple Commands.

1876 **3.162 FIFO Special File (or FIFO)** |

1877 A type of file with the property that data written to such a file is read on a first-in-first-out basis. |

1878 **Note:** Other characteristics of FIFOs are described in the System Interfaces volume of
1879 IEEE Std 1003.1-200x, *lseek()*, *open()*, *read()*, and *write()*.

1880 **3.163 File** |

1881 An object that can be written to, or read from, or both. A file has certain attributes, including |
1882 access permissions and type. File types include regular file, character special file, block special |
1883 file, FIFO special file, symbolic link, socket, and directory. Other types of files may be supported |
1884 by the implementation.

1885 **3.164 File Description**

1886 See *Open File Description* in Section 3.253 (on page 67). |

1887 **3.165 File Descriptor** |

1888 A per-process unique, non-negative integer used to identify an open file for the purpose of file |
1889 access. The value of a file descriptor is from zero to {OPEN_MAX}. A process can have no more |
1890 than {OPEN_MAX} file descriptors open simultaneously. File descriptors may also be used to |
1891 implement message catalog descriptors and directory streams; see also Section 3.253 (on page |
1892 67).

1893 **Note:** {OPEN_MAX} is defined in detail in <limits.h>.

1894 **3.166 File Group Class** |

1895 The property of a file indicating access permissions for a process related to the group |
1896 identification of a process. A process is in the file group class of a file if the process is not in the |
1897 file owner class and if the effective group ID or one of the supplementary group IDs of the |
1898 process matches the group ID associated with the file. Other members of the class may be |
1899 implementation-defined.

1900 **3.167 File Mode** |

1901 An object containing the *file mode bits* and file type of a file. |

1902 **Note:** File mode bits and file types are defined in detail in <sys/stat.h>.

1903 3.168 File Mode Bits

1904 A file's file permission bits, set-user-ID-on-execution bit (S_ISUID), and set-group-ID-on-
1905 execution bit (S_ISGID).

1906 **Note:** File Mode Bits are defined in detail in <sys/stat.h>.

1907 3.169 Filename

1908 A name consisting of 1 to {NAME_MAX} bytes used to name a file. The characters composing
1909 the name may be selected from the set of all character values excluding the slash character and
1910 the null byte. The filenames dot and dot-dot have special meaning. A filename is sometimes
1911 referred to as a *pathname component*.

1912 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 98).

1913 3.170 Filename Portability

1914 Filenames should be constructed from the portable filename character set because the use of
1915 other characters can be confusing or ambiguous in certain contexts. (For example, the use of a
1916 colon (':') in a pathname could cause ambiguity if that pathname were included in a *PATH*
1917 definition.)

1918 3.171 File Offset

1919 The byte position in the file where the next I/O operation begins. Each open file description
1920 associated with a regular file, block special file, or directory has a file offset. A character special
1921 file that does not refer to a terminal device may have a file offset. There is no file offset specified
1922 for a pipe or FIFO.

1923 3.172 File Other Class

1924 The property of a file indicating access permissions for a process related to the user and group
1925 identification of a process. A process is in the file other class of a file if the process is not in the
1926 file owner class or file group class.

1927 3.173 File Owner Class

1928 The property of a file indicating access permissions for a process related to the user
1929 identification of a process. A process is in the file owner class of a file if the effective user ID of
1930 the process matches the user ID of the file.

1931 3.174 File Permission Bits

1932 Information about a file that is used, along with other information, to determine whether a
1933 process has read, write, or execute/search permission to a file. The bits are divided into three
1934 parts: owner, group, and other. Each part is used with the corresponding file class of processes.
1935 These bits are contained in the file mode.

1936 **Note:** File modes are defined in detail in `<sys/stat.h>`.

1937 File Access Permissions are defined in detail in Section 4.4 (on page 95).

1938 **3.175 File Serial Number**

1939 A per-file system unique identifier for a file.

1940 **3.176 File System**

1941 A collection of files and certain of their attributes. It provides a name space for file serial
1942 numbers referring to those files.

1943 **3.177 File Type**

1944 See *File* in Section 3.163 (on page 55).

1945 **3.178 Filter**

1946 A command whose operation consists of reading data from standard input or a list of input files
1947 and writing data to standard output. Typically, its function is to perform some transformation
1948 on the data stream.

1949 **3.179 First Open (of a File)**

1950 When a process opens a file that is not currently an open file within any process.

1951 **3.180 Flow Control**

1952 The mechanism employed by a communications provider that constrains a sending entity to
1953 wait until the receiving entities can safely receive additional data without loss.

1954 **3.181 Foreground Job**

1955 See *Foreground Process Group* in Section 3.183.

1956 **3.182 Foreground Process**

1957 A process that is a member of a foreground process group. |

1958 **3.183 Foreground Process Group (or Foreground Job)** |

1959 A process group whose member processes have certain privileges, denied to processes in |
1960 background process groups, when accessing their controlling terminal. Each session that has |

1961 established a connection with a controlling terminal has at most one process group of the session
1962 as the foreground process group of that controlling terminal.

1963 **Note:** The General Terminal Interface is defined in detail in Chapter 11.

1964 **3.184 Foreground Process Group ID**

1965 The process group ID of the foreground process group. |

1966 **3.185 Form-Feed Character (<form-feed>)**

1967 A character that in the output stream indicates that printing should start on the next page of an
1968 output device. It is the character designated by '`\f`' in the C language. If the <form-feed> is not
1969 the first character of an output line, the result is unspecified. It is unspecified whether this
1970 character is the exact sequence transmitted to an output device by the system to accomplish the
1971 movement to the next page. |

1972 **3.186 Graphic Character**

1973 A member of the **graph** character class of the current locale. |

1974 **Note:** The **graph** character class is defined in detail in Section 7.3.1 (on page 122).

1975 **3.187 Group Database**

1976 A system database of implementation-defined format that contains at least the following
1977 information for each group ID: |

- 1978 • Group name
- 1979 • Numerical group ID
- 1980 • List of users allowed in the group

1981 The list of users allowed in the group is used by the *newgrp* utility.

1982 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x.

1983 **3.188 Group ID**

1984 A non-negative integer, which can be contained in an object of type **gid_t**, that is used to identify
1985 a group of system users. Each system user is a member of at least one group. When the identity
1986 of a group is associated with a process, a group ID value is referred to as a real group ID, an
1987 effective group ID, one of the supplementary group IDs, or a saved set-group-ID. |

1988 **3.189 Group Name**

1989 A string that is used to identify a group; see also Section 3.187. To be portable across conforming
1990 systems, the value is composed of characters from the portable filename character set. The
1991 hyphen should not be used as the first character of a portable group name.

1992 3.190 Hard Limit

1993 A system resource limitation that may be reset to a lesser or greater limit by a privileged process.
1994 A non-privileged process is restricted to only lowering its hard limit. |

1995 3.191 Hard Link

1996 The relationship between two directory entries that represent the same file; see also Section 3.129 |
1997 (on page 50). The result of an execution of the *ln* utility (without the *-s* option) or the *link()* |
1998 function. This term is contrasted against symbolic link; see also Section 3.372 (on page 83). |

1999 3.192 Home Directory

2000 The directory specified by the *HOME* environment variable. |

2001 3.193 Host Byte Order

2002 The arrangement of bytes in any **int** type when using a specific machine architecture. |

2003 **Note:** Two common methods of byte ordering are big-endian and little-endian. Big-endian is a
2004 format for storage of binary data in which the most significant byte is placed first, with the rest
2005 in descending order. Little-endian is a format for storage or transmission of binary data in
2006 which the least significant byte is placed first, with the rest in ascending order. See also Section |
2007 4.8 (on page 97). |

2008 3.194 Incomplete Line

2009 A sequence of one or more non-*<newline>*s at the end of the file.

2010 3.195 Inf

2011 A value representing +infinity or a value representing -infinity that can be stored in a floating
2012 type. Not all systems support the Inf values.

2013 3.196 Instrumented Application

2014 An application that contains at least one call to the trace point function *posix_trace_event()*. Each
2015 process of an instrumented application has a mapping of trace event names to trace event type
2016 identifiers. This mapping is used by the trace stream that is created for that process. |

2017 3.197 Interactive Shell

2018 A processing mode of the shell that is suitable for direct user interaction. |

2019 **3.198 Internationalization**

2020 The provision within a computer program of the capability of making itself adaptable to the
2021 requirements of different native languages, local customs, and coded character sets.

2022 **3.199 Interprocess Communication**

2023 A functionality enhancement to add a high-performance, deterministic interprocess
2024 communication facility for local communication.

2025 **3.200 Invoke**

2026 To perform command search and execution actions, except that searching for shell functions and
2027 special built-in utilities is suppressed; see also Section 3.154 (on page 53).

2028 **Note:** Command Search and Execution is defined in detail in the Shell and Utilities volume of
2029 IEEE Std 1003.1-200x, Section 2.9.1.1, Command Search and Execution.

2030 **3.201 Job**

2031 A set of processes, comprising a shell pipeline, and any processes descended from it, that are all
2032 in the same process group.

2033 **Note:** See also the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.9.2, Pipelines.

2034 **3.202 Job Control**

2035 A facility that allows users selectively to stop (suspend) the execution of processes and continue
2036 (resume) their execution at a later point. The user typically employs this facility via the
2037 interactive interface jointly supplied by the terminal I/O driver and a command interpreter.

2038 **3.203 Job Control Job ID**

2039 A handle that is used to refer to a job. The job control job ID can be any of the forms shown in the
2040 following table:

2041

Table 3-1 Job Control Job ID Formats

2042

2043

2044

2045

2046

2047

2048

2049

Job Control Job ID	Meaning
%%	Current job.
%+	Current job.
%-	Previous job.
% <i>n</i>	Job number <i>n</i> .
% <i>string</i>	Job whose command begins with <i>string</i> .
%? <i>string</i>	Job whose command contains <i>string</i> .

2050 **3.204 Last Close (of a File)**

2051 When a process closes a file, resulting in the file not being an open file within any process.

2052 **3.205 Line**

2053 A sequence of zero or more non-<newline>s plus a terminating <newline>.

2054 **3.206 Linger**

2055 A period of time before terminating a connection, to allow outstanding data to be transferred.

2056 **3.207 Link**2057 See *Directory Entry* in Section 3.129 (on page 50).2058 **3.208 Link Count**

2059 The number of directory entries that refer to a particular file.

2060 **3.209 Local Customs**2061 The conventions of a geographical area or territory for such things as date, time, and currency
2062 formats.2063 **3.210 Local Interprocess Communication (Local IPC)**

2064 The transfer of data between processes in the same system. |

2065 **3.211 Locale** |2066 The definition of the subset of a user's environment that depends on language and cultural
2067 conventions. |

2068 **Note:** Locales are defined in detail in Chapter 7 (on page 119).

2069 **3.212 Localization**

2070 The process of establishing information within a computer system specific to the operation of
2071 particular native languages, local customs, and coded character sets.

2072 **3.213 Login**

2073 The unspecified activity by which a user gains access to the system. Each login is associated
2074 with exactly one login name.

2075 **3.214 Login Name**

2076 A user name that is associated with a login.

2077 **3.215 Map**

2078 To create an association between a page-aligned range of the address space of a process and
2079 some memory object, such that a reference to an address in that range of the address space
2080 results in a reference to the associated memory object. The mapped memory object is not
2081 necessarily memory-resident. |

2082 **3.216 Marked Message**

2083 A STREAMS message on which a certain flag is set. Marking a message gives the application |
2084 protocol-specific information. An application can use *ioctl()* to determine whether a given |
2085 message is marked.

2086 **Note:** The *ioctl()* function is defined in detail in the System Interfaces volume of
2087 IEEE Std 1003.1-200x.

2088 **3.217 Matched**

2089 A state applying to a sequence of zero or more characters when the characters in the sequence |
2090 correspond to a sequence of characters defined by a basic regular expression or extended regular |
2091 expression pattern.

2092 **Note:** Regular Expressions are defined in detail in Chapter 9 (on page 165).

2093 **3.218 Memory Mapped Files**

2094 A facility to allow applications to access files as part of the address space. |

2095 3.219 Memory Object |

2096 One of: |

- 2097 • A file (see Section 3.163 (on page 55))
- 2098 • A shared memory object (see Section 3.340 (on page 79))
- 2099 • A typed memory object (see Section 3.418 (on page 90))

2100 When used in conjunction with *mmap()*, a memory object appears in the address space of the
2101 calling process.

2102 **Note:** The *mmap()* function is defined in detail in the System Interfaces volume of
2103 IEEE Std 1003.1-200x.

2104 3.220 Memory-Resident

2105 The process of managing the implementation in such a way as to provide an upper bound on
2106 memory access times.

2107 3.221 Message

2108 In the context of programmatic message passing, information that can be transferred between
2109 processes or threads by being added to and removed from a message queue. A message consists
2110 of a fixed-size message buffer.

2111 3.222 Message Catalog

2112 In the context of providing natural language messages to the user, a file or storage area
2113 containing program messages, command prompts, and responses to prompts for a particular
2114 native language, territory, and codeset.

2115 3.223 Message Catalog Descriptor

2116 In the context of providing natural language messages to the user, a per-process unique value
2117 used to identify an open message catalog. A message catalog descriptor may be implemented
2118 using a file descriptor.

2119 3.224 Message Queue

2120 In the context of programmatic message passing, an object to which messages can be added and
2121 removed. Messages may be removed in the order in which they were added or in priority order. |

2122 3.225 Mode |

2123 A collection of attributes that specifies a file's type and its access permissions. |

2124 **Note:** File Access Permissions are defined in detail in Section 4.4 (on page 95).

2125 3.226 Monotonic Clock

2126 A clock whose value cannot be set via `clock_settime()` and which cannot have negative clock
2127 jumps.

2128 3.227 Mount Point

2129 Either the system root directory or a directory for which the `st_dev` field of structure `stat` differs
2130 from that of its parent directory.

2131 **Note:** The `stat` structure is defined in detail in `<sys/stat.h>`.

2132 3.228 Multi-Character Collating Element

2133 A sequence of two or more characters that collate as an entity. For example, in some coded
2134 character sets, an accented character is represented by a non-spacing accent, followed by the
2135 letter. Other examples are the Spanish elements *ch* and *ll*.

2136 3.229 Mutex

2137 A synchronization object used to allow multiple threads to serialize their access to shared data.
2138 The name derives from the capability it provides; namely, mutual-exclusion. The thread that has
2139 locked a mutex becomes its owner and remains the owner until that same thread unlocks the
2140 mutex.

2141 3.230 Name

2142 In the shell command language, a word consisting solely of underscores, digits, and alphabets
2143 from the portable character set. The first character of a name is not a digit.

2144 **Note:** The portable character set is defined in detail in Section 6.1 (on page 111).

2145 3.231 Named STREAM

2146 A STREAMS-based file descriptor that is attached to a name in the file system name space. All
2147 subsequent operations on the named STREAM act on the STREAM that was associated with the
2148 file descriptor until the name is disassociated from the STREAM.

2149 3.232 NaN (Not a Number)

2150 A set of values that may be stored in a floating type but that are neither Inf nor valid floating-
2151 point numbers. Not all systems support NaN values.

2152 3.233 Native Language

2153 A computer user's spoken or written language, such as American English, British English,
2154 Danish, Dutch, French, German, Italian, Japanese, Norwegian, or Swedish.

2155 3.234 Negative Response |

2156 An input string that matches one of the responses acceptable to the *LC_MESSAGES* category |
2157 keyword **noexpr**, matching an extended regular expression in the current locale. |

2158 **Note:** The *LC_MESSAGES* category is defined in detail in Section 7.3.6 (on page 148). |

2159 3.235 Network |

2160 A collection of interconnected hosts. |

2161 **Note:** The term network in IEEE Std 1003.1-200x is used to refer to the network of hosts. The term |
2162 batch system is used to refer to the network of batch servers. |

2163 3.236 Network Address |

2164 A network-visible identifier used to designate specific endpoints in a network. Specific |
2165 endpoints on host systems have addresses, and host systems may also have addresses. |

2166 3.237 Network Byte Order |

2167 The way of representing any **int** type such that, when transmitted over a network via a network |
2168 endpoint, the **int** type is transmitted as an appropriate number of octets with the most |
2169 significant octet first, followed by any other octets in descending order of significance. |

2170 **Note:** This order is more commonly known as big-endian ordering. See also Section 4.8 (on page 97). |

2171 3.238 Newline Character (<newline>) |

2172 A character that in the output stream indicates that printing should start at the beginning of the |
2173 next line. It is the character designated by '`\n`' in the C language. It is unspecified whether this |
2174 character is the exact sequence transmitted to an output device by the system to accomplish the |
2175 movement to the next line. |

2176 3.239 Nice Value |

2177 A number used as advice to the system to alter process scheduling. Numerically smaller values |
2178 give a process additional preference when scheduling a process to run. Numerically larger |
2179 values reduce the preference and make a process less likely to run. Typically, a process with a |
2180 smaller nice value runs to completion more quickly than an equivalent process with a higher |
2181 nice value. The symbol {NZERO} specifies the default nice value of the system. |

2182 3.240 Non-Blocking |

2183 A property of an open file description that causes function calls involving it to return without |
2184 delay when it is detected that the requested action associated with the function call cannot be |
2185 completed without unknown delay. |

2186 **Note:** The exact semantics are dependent on the type of file associated with the open file description. |
2187 For data reads from devices such as ttys and FIFOs, this property causes the read to return |

2188 immediately when no data was available. Similarly, for writes, it causes the call to return |
2189 immediately when the thread would otherwise be delayed in the write operation; for example, |
2190 because no space was available. For networking, it causes functions not to await protocol |
2191 events (for example, acknowledgements) to occur. See also the System Interfaces volume of |
2192 IEEE Std 1003.1-200x, Section 2.10.7, Socket I/O Mode. |

2193 **3.241 Non-Spacing Characters**

2194 A character, such as a character representing a diacritical mark in the ISO/IEC 6937:1994
2195 standard coded character set, which is used in combination with other characters to form
2196 composite graphic symbols.

2197 **3.242 NUL**

2198 A character with all bits set to zero.

2199 **3.243 Null Byte**

2200 A byte with all bits set to zero.

2201 **3.244 Null Pointer**

2202 The value that is obtained by converting the number 0 into a pointer; for example, (**void ***) 0. The
2203 C language guarantees that this value does not match that of any legitimate pointer, so it is used
2204 by many functions that return pointers to indicate an error.

2205 **3.245 Null String**

2206 See *Empty String* in Section 3.145 (on page 52).

2207 **3.246 Null Wide-Character Code**

2208 A wide-character code with all bits set to zero.

2209 **3.247 Number Sign**

2210 The character ' # ', also known as *hash sign*. |

2211 **3.248 Object File**

2212 A regular file containing the output of a compiler, formatted as input to a linkage editor for |
2213 linking with other object files into an executable form. The methods of linking are unspecified |
2214 and may involve the dynamic linking of objects at runtime. The internal format of an object file |
2215 is unspecified, but a conforming application cannot assume an object file is a text file. |

2216 3.249 Octet

2217 Unit of data representation that consists of eight contiguous bits.

2218 3.250 Offset Maximum

2219 An attribute of an open file description representing the largest value that can be used as a file
2220 offset.

2221 3.251 Opaque Address

2222 An address such that the entity making use of it requires no details about its contents or format.

2223 3.252 Open File

2224 A file that is currently associated with a file descriptor. |

2225 3.253 Open File Description |

2226 A record of how a process or group of processes is accessing a file. Each file descriptor refers to |
2227 exactly one open file description, but an open file description can be referred to by more than |
2228 one file descriptor. A file offset, file status, and file access modes are attributes of an open file |
2229 description. |

2230 3.254 Operand |

2231 An argument to a command that is generally used as an object supplying information to a utility |
2232 necessary to complete its processing. Operands generally follow the options in a command line. |

2233 **Note:** Utility Argument Syntax is defined in detail in Section 12.1 (on page 197).

2234 3.255 Operator |

2235 In the shell command language, either a control operator or a redirection operator. |

2236 3.256 Option |

2237 An argument to a command that is generally used to specify changes in the utility's default |
2238 behavior. |

2239 **Note:** Utility Argument Syntax is defined in detail in Section 12.1 (on page 197).

2240 3.257 Option-Argument |

2241 A parameter that follows certain options. In some cases an option-argument is included within |
2242 the same argument string as the option—in most cases it is the next argument. |

2243 **Note:** Utility Argument Syntax is defined in detail in Section 12.1 (on page 197).

2244 **3.258 Orientation**

2245 A stream has one of three orientations: unoriented, byte-oriented, or wide-oriented.

2246 **Note:** For further information, see the System Interfaces volume of IEEE Std 1003.1-200x, Section
2247 2.5.2, Stream Orientation and Encoding Rules.

2248 **3.259 Orphaned Process Group**

2249 A process group in which the parent of every member is either itself a member of the group or is
2250 not a member of the group's session.

2251 **3.260 Page**

2252 The granularity of process memory mapping or locking.

2253 Physical memory and memory objects can be mapped into the address space of a process on
2254 page boundaries and in integral multiples of pages. Process address space can be locked into
2255 memory (made memory-resident) on page boundaries and in integral multiples of pages.

2256 **3.261 Page Size**

2257 The size, in bytes, of the system unit of memory allocation, protection, and mapping. On systems
2258 that have segment rather than page-based memory architectures, the term page means a
2259 segment.

2260 **3.262 Parameter**

2261 In the shell command language, an entity that stores values. There are three types of parameters:
2262 variables (named parameters), positional parameters, and special parameters. Parameter
2263 expansion is accomplished by introducing a parameter with the '\$' character.

2264 **Note:** See also the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.5, Parameters and
2265 Variables.

2266 In the C language, an object declared as part of a function declaration or definition that acquires
2267 a value on entry to the function, or an identifier following the macro name in a function-like
2268 macro definition.

2269 **3.263 Parent Directory**

2270 When discussing a given directory, the directory that both contains a directory entry for the
2271 given directory and is represented by the pathname dot-dot in the given directory.

2272 When discussing other types of files, a directory containing a directory entry for the file under
2273 discussion.

2274 This concept does not apply to dot and dot-dot.

2275 **3.264 Parent Process**

2276 The process which created (or inherited) the process under discussion.

2277 **3.265 Parent Process ID**

2278 An attribute of a new process identifying the parent of the process. The parent process ID of a
2279 process is the process ID of its creator, for the lifetime of the creator. After the creator's lifetime
2280 has ended, the parent process ID is the process ID of an implementation-defined system process. |

2281 **3.266 Pathname**

2282 A character string that is used to identify a file. In the context of IEEE Std 1003.1-200x, a |
2283 pathname consists of, at most, {PATH_MAX} bytes, including the terminating null byte. It has an |
2284 optional beginning slash, followed by zero or more filenames separated by slashes. A pathname
2285 may optionally contain one or more trailing slashes. Multiple successive slashes are considered
2286 to be the same as one slash.

2287 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 98).

2288 **3.267 Pathname Component**

2289 See *Filename* in Section 3.169 (on page 56).

2290 **3.268 Path Prefix**

2291 A pathname, with an optional ending slash, that refers to a directory. |

2292 **3.269 Pattern**

2293 A sequence of characters used either with regular expression notation or for pathname |
2294 expansion, as a means of selecting various character strings or pathnames, respectively. |

2295 **Note:** Regular Expressions are defined in detail in Chapter 9 (on page 165).

2296 See also the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.6.6, Pathname
2297 Expansion.

2298 The syntaxes of the two types of patterns are similar, but not identical; IEEE Std 1003.1-200x
2299 always indicates the type of pattern being referred to in the immediate context of the use of the
2300 term.

2301 **3.270 Period**

2302 The character ' . '. The term period is contrasted with dot (see also Section 3.135 (on page 51)),
2303 which is used to describe a specific directory entry. |

2304 **3.271 Permissions** |

2305 Attributes of an object that determine the privilege necessary to access or manipulate the object. |

2306 **Note:** File Access Permissions are defined in detail in Section 4.4 (on page 95). |

2307 **3.272 Persistence** |

2308 A mode for semaphores, shared memory, and message queues requiring that the object and its |
2309 state (including data, if any) are preserved after the object is no longer referenced by any process. |

2310 Persistence of an object does not imply that the state of the object is maintained across a system |
2311 crash or a system reboot. |

2312 **3.273 Pipe** |

2313 An object accessed by one of the pair of file descriptors created by the *pipe()* function. Once |
2314 created, the file descriptors can be used to manipulate it, and it behaves identically to a FIFO |
2315 special file when accessed in this way. It has no name in the file hierarchy. |

2316 **Note:** The *pipe()* function is defined in detail in the System Interfaces volume of |
2317 IEEE Std 1003.1-200x. |

2318 **3.274 Polling** |

2319 A scheduling scheme whereby the local process periodically checks until the prespecified events |
2320 (for example, read, write) have occurred. |

2321 **3.275 Portable Character Set** |

2322 The collection of characters that are required to be present in all locales supported by |
2323 conforming systems. |

2324 **Note:** The portable character set is defined in detail in Section 6.1 (on page 111). |

2325 This term is contrasted against the smaller *portable filename character set*; see also Section 3.276. |

2326 **3.276 Portable Filename Character Set** |

2327 The set of characters from which portable filenames are constructed. |

2328 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
2329 a b c d e f g h i j k l m n o p q r s t u v w x y z
2330 0 1 2 3 4 5 6 7 8 9 . _ -

2331 The last three characters are the period, underscore, and hyphen characters, respectively. |

2332 **3.277 Positional Parameter** |

2333 In the shell command language, a parameter denoted by a single digit or one or more digits in |
2334 curly braces. |

2335 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section
2336 2.5.1, Positional Parameters.

2337 **3.278 Preallocation**

2338 The reservation of resources in a system for a particular use.

2339 Preallocation does not imply that the resources are immediately allocated to that use, but merely
2340 indicates that they are guaranteed to be available in bounded time when needed.

2341 **3.279 Preempted Process (or Thread)**

2342 A running thread whose execution is suspended due to another thread becoming runnable at a
2343 higher priority.

2344 **3.280 Previous Job**

2345 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities if the
2346 current job exits. There is at most one previous job; see also Section 3.203 (on page 60).

2347 **3.281 Printable Character**

2348 One of the characters included in the **print** character classification of the *LC_CTYPE* category in
2349 the current locale.

2350 **Note:** The *LC_CTYPE* category is defined in detail in Section 7.3.1 (on page 122).

2351 **3.282 Printable File**

2352 A text file consisting only of the characters included in the **print** and **space** character
2353 classifications of the *LC_CTYPE* category and the <backspace>, all in the current locale.

2354 **Note:** The *LC_CTYPE* category is defined in detail in Section 7.3.1 (on page 122).

2355 **3.283 Priority**

2356 A non-negative integer associated with processes or threads whose value is constrained to a
2357 range defined by the applicable scheduling policy. Numerically higher values represent higher
2358 priorities.

2359 **3.284 Priority Band**

2360 The queuing order applied to normal priority STREAMS messages. High priority STREAMS
2361 messages are not grouped by priority bands. The only differentiation made by the STREAMS
2362 mechanism is between zero and non-zero bands, but specific protocol modules may differentiate
2363 between priority bands.

2364 **3.285 Priority Inversion**

2365 A condition in which a thread that is not voluntarily suspended (waiting for an event or time
2366 delay) is not running while a lower priority thread is running. Such blocking of the higher
2367 priority thread is often caused by contention for a shared resource.

2368 **3.286 Priority Scheduling**

2369 A performance and determinism improvement facility to allow applications to determine the
2370 order in which threads that are ready to run are granted access to processor resources.

2371 **3.287 Priority-Based Scheduling**

2372 Scheduling in which the selection of a running thread is determined by the priorities of the
2373 runnable processes or threads.

2374 **3.288 Privilege**

2375 See *Appropriate Privileges* in Section 3.19 (on page 35). |

2376 **3.289 Process** |

2377 An address space with one or more threads executing within that address space, and the |
2378 required system resources for those threads.

2379 **Note:** Many of the system resources defined by IEEE Std 1003.1-200x are shared among all of the
2380 threads within a process. These include the process ID, the parent process ID, process group ID,
2381 session membership, real, effective, and saved-set user ID, real, effective, and saved-set group
2382 ID, supplementary group IDs, current working directory, root directory, file mode creation
2383 mask, and file descriptors.

2384 **3.290 Process Group**

2385 A collection of processes that permits the signaling of related processes. Each process in the
2386 system is a member of a process group that is identified by a process group ID. A newly created
2387 process joins the process group of its creator. |

2388 **3.291 Process Group ID** |

2389 The unique positive integer identifier representing a process group during its lifetime. |

2390 **Note:** See also Process Group ID Reuse defined in Section 4.12 (on page 99).

2391 **3.292 Process Group Leader**

2392 A process whose process ID is the same as its process group ID. |

2393 3.293 Process Group Lifetime |

2394 A period of time that begins when a process group is created and ends when the last remaining
2395 process in the group leaves the group, due either to the end of the last process' lifetime or to the
2396 last remaining process calling the *setsid()* or *setpgid()* functions.

2397 **Note:** The *setsid()* and *setpgid()* functions are defined in detail in the System Interfaces volume of
2398 IEEE Std 1003.1-200x.

2399 3.294 Process ID |

2400 The unique positive integer identifier representing a process during its lifetime. |

2401 **Note:** See also Process ID Reuse defined in Section 4.12 (on page 99).

2402 3.295 Process Lifetime |

2403 The period of time that begins when a process is created and ends when its process ID is
2404 returned to the system. After a process is created with a *fork()* function, it is considered active.
2405 At least one thread of control and address space exist until it terminates. It then enters an
2406 inactive state where certain resources may be returned to the system, although some resources,
2407 such as the process ID, are still in use. When another process executes a *wait()*, *waitid()*, or
2408 *waitpid()* function for an inactive process, the remaining resources are returned to the system.
2409 The last resource to be returned to the system is the process ID. At this time, the lifetime of the
2410 process ends.

2411 **Note:** The *fork()*, *wait()*, *waitid()*, and *waitpid()* functions are defined in detail in the System
2412 Interfaces volume of IEEE Std 1003.1-200x.

2413 3.296 Process Memory Locking

2414 A performance improvement facility to bind application programs into the high-performance
2415 random access memory of a computer system. This avoids potential latencies introduced by the
2416 operating system in storing parts of a program that were not recently referenced on secondary
2417 memory devices. |

2418 3.297 Process Termination |

2419 There are two kinds of process termination: |

2420 1. Normal termination occurs by a return from *main()* or when requested with the *exit()* or
2421 *_exit()* functions.

2422 2. Abnormal termination occurs when requested by the *abort()* function or when some
2423 signals are received.

2424 **Note:** The *_exit()*, *abort()*, and *exit()* functions are defined in detail in the System Interfaces volume
2425 of IEEE Std 1003.1-200x.

2426 3.298 Process-To-Process Communication

2427 The transfer of data between processes.

2428 3.299 Process Virtual Time

2429 The measurement of time in units elapsed by the system clock while a process is executing.

2430 3.300 Program

2431 A prepared sequence of instructions to the system to accomplish a defined task. The term
2432 program in IEEE Std 1003.1-200x encompasses applications written in the Shell Command
2433 Language, complex utility input languages (for example, *awk*, *lex*, *sed*, and so on), and high-level
2434 languages.

2435 3.301 Protocol

2436 A set of semantic and syntactic rules for exchanging information.

2437 3.302 Pseudo-Terminal

2438 A facility that provides an interface that is identical to the terminal subsystem. A pseudo-
2439 terminal is composed of two devices: the *master device* and a *slave device*. The slave device
2440 provides processes with an interface that is identical to the terminal interface, although there
2441 need not be hardware behind that interface. Anything written on the master device is presented
2442 to the slave as an input and anything written on the slave device is presented as an input on the
2443 master side.

2444 3.303 Radix Character

2445 The character that separates the integer part of a number from the fractional part.

2446 3.304 Read-Only File System

2447 A file system that has implementation-defined characteristics restricting modifications.

2448 **Note:** File Times Update is described in detail in Section 4.7 (on page 96).

2449 3.305 Read-Write Lock

2450 Multiple readers, single writer (read-write) locks allow many threads to have simultaneous
2451 read-only access to data while allowing only one thread to have write access at any given time.
2452 They are typically used to protect data that is read-only more frequently than it is changed.

2453 Read-write locks can be used to synchronize threads in the current process and other processes if
2454 they are allocated in memory that is writable and shared among the cooperating processes and
2455 have been initialized for this behavior.

2456 **3.306 Real Group ID**

2457 The attribute of a process that, at the time of process creation, identifies the group of the user
2458 who created the process; see also Section 3.188 (on page 58).

2459 **3.307 Real Time**

2460 Time measured as total units elapsed by the system clock without regard to which thread is
2461 executing.

2462 **3.308 Realtime Signal Extension**

2463 A determinism improvement facility to enable asynchronous signal notifications to an
2464 application to be queued without impacting compatibility with the existing signal functions.

2465 **3.309 Real User ID**

2466 The attribute of a process that, at the time of process creation, identifies the user who created the
2467 process; see also Section 3.425 (on page 91).

2468 **3.310 Record**

2469 A collection of related data units or words which is treated as a unit. |

2470 **3.311 Redirection** |

2471 In the shell command language, a method of associating files with the input or output of |
2472 commands.

2473 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.7,
2474 Redirection.

2475 **3.312 Redirection Operator**

2476 In the shell command language, a token that performs a redirection function. It is one of the
2477 following symbols:

2478 < > >| << >> <& >& <<- <>

2479 **3.313 Reentrant Function**

2480 A function whose effect, when called by two or more threads, is guaranteed to be as if the
2481 threads each executed the function one after another in an undefined order, even if the actual
2482 execution is interleaved.

2483 3.314 Referenced Shared Memory Object

2484 A shared memory object that is open or has one or more mappings defined on it.

2485 3.315 Refresh

2486 To ensure that the information on the user's terminal screen is up-to-date. |

2487 3.316 Regular Expression |

2488 A pattern that selects specific strings from a set of character strings. |

2489 **Note:** Regular Expressions are described in detail in Chapter 9 (on page 165).

2490 3.317 Region |

2491 In the context of the address space of a process, a sequence of addresses. |

2492 In the context of a file, a sequence of offsets.

2493 3.318 Regular File

2494 A file that is a randomly accessible sequence of bytes, with no further structure imposed by the
2495 system. |

2496 3.319 Relative Pathname |

2497 A pathname not beginning with a slash. |

2498 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 98).

2499 3.320 Relocatable File

2500 A file holding code or data suitable for linking with other object files to create an executable or a
2501 shared object file.

2502 3.321 Relocation

2503 The process of connecting symbolic references with symbolic definitions. For example, when a
2504 program calls a function, the associated call instruction transfers control to the proper
2505 destination address at execution.

2506 3.322 Requested Batch Service

2507 A service that is either rejected or performed prior to a response from the service to the
2508 requester.

2509 3.323 (Time) Resolution

2510 The minimum time interval that a clock can measure or whose passage a timer can detect.

2511 3.324 Root Directory

2512 A directory, associated with a process, that is used in pathname resolution for pathnames that
2513 begin with a slash.

2514 3.325 Runnable Process (or Thread)

2515 A thread that is capable of being a running thread, but for which no processor is available.

2516 3.326 Running Process (or Thread)

2517 A thread currently executing on a processor. On multi-processor systems there may be more
2518 than one such thread in a system at a time.

2519 3.327 Saved Resource Limits

2520 An attribute of a process that provides some flexibility in the handling of unrepresentable
2521 resource limits, as described in the *exec* family of functions and *setrlimit()*.

2522 **Note:** The *exec* and *setrlimit()* functions are defined in detail in the System Interfaces volume of
2523 IEEE Std 1003.1-200x.

2524 3.328 Saved Set-Group-ID

2525 An attribute of a process that allows some flexibility in the assignment of the effective group ID
2526 attribute, as described in the *exec* family of functions and *setgid()*.

2527 **Note:** The *exec* and *setgid()* functions are defined in detail in the System Interfaces volume of
2528 IEEE Std 1003.1-200x.

2529 3.329 Saved Set-User-ID

2530 An attribute of a process that allows some flexibility in the assignment of the effective user ID
2531 attribute, as described in the *exec* family of functions and *setuid()*.

2532 **Note:** The *exec* and *setuid()* functions are defined in detail in the System Interfaces volume of
2533 IEEE Std 1003.1-200x.

2534 3.330 Scheduling

2535 The application of a policy to select a runnable process or thread to become a running process or
2536 thread, or to alter one or more of the thread lists.

2537 3.331 Scheduling Allocation Domain

2538 The set of processors on which an individual thread can be scheduled at any given time. |

2539 3.332 Scheduling Contention Scope |

2540 A property of a thread that defines the set of threads against which that thread competes for |
2541 resources.

2542 For example, in a scheduling decision, threads sharing scheduling contention scope compete for
2543 processor resources. In IEEE Std 1003.1-200x, a thread has scheduling contention scope of either
2544 PTHREAD_SCOPE_SYSTEM or PTHREAD_SCOPE_PROCESS. |

2545 3.333 Scheduling Policy |

2546 A set of rules that is used to determine the order of execution of processes or threads to achieve |
2547 some goal.

2548 **Note:** Scheduling Policy is defined in detail in Section 4.13 (on page 99).

2549 3.334 Screen

2550 A rectangular region of columns and lines on a terminal display. A screen may be a portion of a |
2551 physical display device or may occupy the entire physical area of the display device.

2552 3.335 Scroll |

2553 To move the representation of data vertically or horizontally relative to the terminal screen. |
2554 There are two types of scrolling:

- 2555 1. The cursor moves with the data.
- 2556 2. The cursor remains stationary while the data moves.

2557 3.336 Semaphore |

2558 A minimum synchronization primitive to serve as a basis for more complex synchronization |
2559 mechanisms to be defined by the application program.

2560 **Note:** Semaphores are defined in detail in Section 4.15 (on page 100).

2561 3.337 Session |

2562 A collection of process groups established for job control purposes. Each process group is a |
2563 member of a session. A process is considered to be a member of the session of which its process
2564 group is a member. A newly created process joins the session of its creator. A process can alter
2565 its session membership; see *setsid()*. There can be multiple process groups in the same session.

2566 **Note:** The *setsid()* function is defined in detail in the System Interfaces volume of
2567 IEEE Std 1003.1-200x.

2568 3.338 Session Leader |

2569 A process that has created a session. |

2570 **Note:** For further information, see the *setsid()* function defined in the System Interfaces volume of
2571 IEEE Std 1003.1-200x.

2572 3.339 Session Lifetime

2573 The period between when a session is created and the end of the lifetime of all the process
2574 groups that remain as members of the session.

2575 3.340 Shared Memory Object

2576 An object that represents memory that can be mapped concurrently into the address space of
2577 more than one process.

2578 3.341 Shell

2579 A program that interprets sequences of text input as commands. It may operate on an input
2580 stream or it may interactively prompt and read commands from a terminal. |

2581 3.342 Shell, the |

2582 The Shell Command Language Interpreter; a specific instance of a shell. |

2583 **Note:** For further information, see the *sh* utility defined in the Shell and Utilities volume of
2584 IEEE Std 1003.1-200x.

2585 3.343 Shell Script |

2586 A file containing shell commands. If the file is made executable, it can be executed by specifying
2587 its name as a simple command. Execution of a shell script causes a shell to execute the
2588 commands within the script. Alternatively, a shell can be requested to execute the commands in
2589 a shell script by specifying the name of the shell script as the operand to the *sh* utility. |

2590 **Note:** Simple Commands are defined in detail in the Shell and Utilities volume of
2591 IEEE Std 1003.1-200x, Section 2.9.1, Simple Commands.

2592 The *sh* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x.

2593 3.344 Signal

2594 A mechanism by which a process or thread may be notified of, or affected by, an event occurring
2595 in the system. Examples of such events include hardware exceptions and specific actions by
2596 processes. The term signal is also used to refer to the event itself.

2597 3.345 Signal Stack

2598 Memory established for a thread, in which signal handlers catching signals sent to that thread
2599 are executed.

2600 3.346 Single-Quote

2601 The character ' ' , also known as *apostrophe*.

2602 3.347 Slash

2603 The character ' / ' , also known as *solidus*.

2604 3.348 Socket

2605 A file of a particular type that is used as a communications endpoint for process-to-process
2606 communication as described in the System Interfaces volume of IEEE Std 1003.1-200x.

2607 3.349 Socket Address

2608 An address associated with a socket or remote endpoint, including an address family identifier
2609 and addressing information specific to that address family. The address may include multiple
2610 parts, such as a network address associated with a host system and an identifier for a specific
2611 endpoint.

2612 3.350 Soft Limit

2613 A resource limitation established for each process that the process may set to any value less than
2614 or equal to the hard limit. |

2615 3.351 Source Code |

2616 When dealing with the Shell Command Language, input to the command language interpreter. |
2617 The term shell script is synonymous with this meaning.

2618 When dealing with an ISO/IEC-conforming programming language, source code is input to a
2619 compiler conforming to that ISO/IEC standard.

2620 Source code also refers to the input statements prepared for the following standard utilities:
2621 *awk, bc, ed, lex, localedef, make, sed, and yacc*.

2622 Source code can also refer to a collection of sources meeting any or all of these meanings.

2623 **Note:** The *awk, bc, ed, lex, localedef, make, sed, and yacc* utilities are defined in detail in the Shell and
2624 Utilities volume of IEEE Std 1003.1-200x.

2625 **3.352 Space Character (<space>)** |

2626 The character defined in the portable character set as <space>. The <space> is a member of the
 2627 **space** character class of the current locale, but represents the single character, and not all of the
 2628 possible members of the class; see also Section 3.431 (on page 92). |

2629 **3.353 Spawn** |

2630 A process creation primitive useful for systems that have difficulty with *fork()* and as an efficient
 2631 replacement for *fork()/exec*. |

2632 **3.354 Special Built-In** |

2633 See *Built-In Utility* in Section 3.83 (on page 44). |

2634 **3.355 Special Parameter** |

2635 In the shell command language, a parameter named by a single character from the following list: |

2636 * @ # ? ! - \$ 0

2637 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section
 2638 2.5.2, Special Parameters. |

2639 **3.356 Spin Lock** |

2640 A synchronization object used to allow multiple threads to serialize their access to shared data. |

2641 **3.357 Sporadic Server** |

2642 A scheduling policy for threads and processes that reserves a certain amount of execution
 2643 capacity for processing aperiodic events at a given priority level. |

2644 **3.358 Standard Error** |

2645 An output stream usually intended to be used for diagnostic messages. |

2646 **3.359 Standard Input** |

2647 An input stream usually intended to be used for primary data input. |

2648 **3.360 Standard Output** |

2649 An output stream usually intended to be used for primary data output. |

2650 **3.361 Standard Utilities**

2651 The utilities described in the Shell and Utilities volume of IEEE Std 1003.1-200x. |

2652 **3.362 Stream**

2653 Appearing in lowercase, a stream is a file access object that allows access to an ordered sequence
 2654 of characters, as described by the ISO C standard. Such objects can be created by the *fdopen()*,
 2655 *fopen()*, or *popen()* functions, and are associated with a file descriptor. A stream provides the
 2656 additional services of user-selectable buffering and formatted input and output; see also Section
 2657 3.363.

2658 **Note:** For further information, see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.5,
 2659 Standard I/O Streams.

2660 The *fdopen()*, *fopen()*, or *popen()* functions are defined in detail in the System Interfaces volume
 2661 of IEEE Std 1003.1-200x.

2662 **3.363 STREAM**

2663 Appearing in uppercase, STREAM refers to a full duplex connection between a process and an
 2664 open device or pseudo-device. It optionally includes one or more intermediate processing
 2665 modules that are interposed between the process end of the STREAM and the device driver (or
 2666 pseudo-device driver) end of the STREAM; see also Section 3.362.

2667 **Note:** For further information, see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.6,
 2668 STREAMS.

2669 **3.364 STREAM End**

2670 The STREAM end is the driver end of the STREAM and is also known as the downstream end of
 2671 the STREAM.

2672 **3.365 STREAM Head**

2673 The STREAM head is the beginning of the STREAM and is at the boundary between the system
 2674 and the application process. This is also known as the upstream end of the STREAM.

2675 **3.366 STREAMS Multiplexor**

2676 A driver with multiple STREAMS connected to it. Multiplexing with STREAMS connected above
 2677 is referred to as N-to-1, or *upper multiplexing*. Multiplexing with STREAMS connected below is
 2678 referred to as 1-to-N or *lower multiplexing*.

2679 **3.367 String**

2680 A contiguous sequence of bytes terminated by and including the first null byte. |

2681 3.368 Subshell |

2682 A shell execution environment, distinguished from the main or current shell execution |
2683 environment.

2684 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.12,
2685 Shell Execution Environment.

2686 3.369 Successfully Transferred |

2687 For a write operation to a regular file, when the system ensures that all data written is readable |
2688 on any subsequent open of the file (even one that follows a system or power failure) in the
2689 absence of a failure of the physical storage medium.

2690 For a read operation, when an image of the data on the physical storage medium is available to
2691 the requesting process.

2692 3.370 Supplementary Group ID

2693 An attribute of a process used in determining file access permissions. A process has up to
2694 {NGROUPS_MAX} supplementary group IDs in addition to the effective group ID. The
2695 supplementary group IDs of a process are set to the supplementary group IDs of the parent
2696 process when the process is created.

2697 3.371 Suspended Job

2698 A job that has received a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal that caused the
2699 process group to stop. A suspended job is a background job, but a background job is not
2700 necessarily a suspended job. |

2701 3.372 Symbolic Link |

2702 A type of file with the property that when the file is encountered during pathname resolution, a |
2703 string stored by the file is used to modify the pathname resolution. The stored string has a length
2704 of {SYMLINK_MAX} bytes or fewer.

2705 **Note:** Pathname Resolution is defined in detail in Section 4.11 (on page 98).

2706 3.373 Synchronized Input and Output

2707 A determinism and robustness improvement mechanism to enhance the data input and output
2708 mechanisms, so that an application can ensure that the data being manipulated is physically
2709 present on secondary mass storage devices.

2710 3.374 Synchronized I/O Completion

2711 The state of an I/O operation that has either been successfully transferred or diagnosed as
2712 unsuccessful. |

2713 3.375 Synchronized I/O Data Integrity Completion |

2714 For read, when the operation has been completed or diagnosed if unsuccessful. The read is |
2715 complete only when an image of the data has been successfully transferred to the requesting |
2716 process. If there were any pending write requests affecting the data to be read at the time that |
2717 the synchronized read operation was requested, these write requests are successfully transferred |
2718 prior to reading the data.

2719 For write, when the operation has been completed or diagnosed if unsuccessful. The write is |
2720 complete only when the data specified in the write request is successfully transferred and all file |
2721 system information required to retrieve the data is successfully transferred.

2722 File attributes that are not necessary for data retrieval (access time, modification time, status |
2723 change time) need not be successfully transferred prior to returning to the calling process.

2724 3.376 Synchronized I/O File Integrity Completion

2725 Identical to a synchronized I/O data integrity completion with the addition that all file attributes |
2726 relative to the I/O operation (including access time, modification time, status change time) are |
2727 successfully transferred prior to returning to the calling process.

2728 3.377 Synchronized I/O Operation

2729 An I/O operation performed on a file that provides the application assurance of the integrity of |
2730 its data and files. |

2731 3.378 Synchronous I/O Operation |

2732 An I/O operation that causes the thread requesting the I/O to be blocked from further use of the |
2733 processor until that I/O operation completes. |

2734 **Note:** A synchronous I/O operation does not imply synchronized I/O data integrity completion or |
2735 synchronized I/O file integrity completion.

2736 3.379 Synchronously-Generated Signal |

2737 A signal that is attributable to a specific thread. |

2738 For example, a thread executing an illegal instruction or touching invalid memory causes a |
2739 synchronously-generated signal. Being synchronous is a property of how the signal was |
2740 generated and not a property of the signal number.

2741 3.380 System

2742 An implementation of IEEE Std 1003.1-200x.

2743 3.381 System Crash

2744 An interval initiated by an unspecified circumstance that causes all processes (possibly other
2745 than special system processes) to be terminated in an undefined manner, after which any
2746 changes to the state and contents of files created or written to by an application prior to the
2747 interval are undefined, except as required elsewhere in IEEE Std 1003.1-200x. |

2748 3.382 System Console |

2749 An implementation-defined device that receives messages sent by the *syslog()* function, and the
2750 *fmtmsg()* function when the MM_CONSOLE flat is set. |

2751 **Note:** The *syslog()* and *fmtmsg()* functions are defined in detail in the System Interfaces volume of |
2752 IEEE Std 1003.1-200x. |

2753 3.383 System Databases |

2754 An implementation provides two system databases. |

2755 The *group database* contains the following information for each group:

- 2756 1. Group name
- 2757 2. Numerical group ID
- 2758 3. List of all users allowed in the group

2759 The *user database* contains the following information for each user:

- 2760 1. User name
- 2761 2. Numerical user ID
- 2762 3. Numerical group ID
- 2763 4. Initial working directory
- 2764 5. Initial user program

2765 If the initial user program field is null, the system default is used. If the initial working directory
2766 field is null, the interpretation of that field is implementation-defined. These databases may
2767 contain other fields that are unspecified by IEEE Std 1003.1-200x.

2768 3.384 System Documentation

2769 All documentation provided with an implementation except for the conformance document.
2770 Electronically distributed documents for an implementation are considered part of the system
2771 documentation.

2772 3.385 System Process

2773 An implementation-defined object, other than a process executing an application, that has a
2774 process ID.

2775 3.386 System Reboot

2776 An implementation-defined sequence of events that may result in the loss of transitory data; that
2777 is, data that is not saved in permanent storage. For example, message queues, shared memory,
2778 semaphores, and processes.

2779 3.387 System Trace Event

2780 A trace event that is generated by the implementation, in response either to a system-initiated
2781 action or to an application-requested action, except for a call to *posix_trace_event()*. When
2782 supported by the implementation, a system-initiated action generates a process-independent
2783 system trace event and an application-requested action generates a process-dependent system
2784 trace event. For a system trace event not defined by IEEE Std 1003.1-200x, the associated trace
2785 event type identifier is derived from the implementation-defined name for this trace event, and
2786 the associated data is of implementation-defined content and length.

2787 3.388 System-Wide

2788 Pertaining to events occurring in all processes existing in an implementation at a given point in
2789 time.

2790 3.389 Tab Character (<tab>)

2791 A character that in the output stream indicates that printing or displaying should start at the
2792 next horizontal tabulation position on the current line. It is the character designated by '`\t`' in
2793 the C language. If the current position is at or past the last defined horizontal tabulation
2794 position, the behavior is unspecified. It is unspecified whether this character is the exact
2795 sequence transmitted to an output device by the system to accomplish the tabulation.

2796 3.390 Terminal (or Terminal Device)

2797 A character special file that obeys the specifications of the general terminal interface.

2798 **Note:** The General Terminal Interface is defined in detail in Chapter 11 (on page 183).

2799 3.391 Text Column

2800 A roughly rectangular block of characters capable of being laid out side-by-side next to other
2801 text columns on an output page or terminal screen. The widths of text columns are measured in
2802 column positions.

2803 3.392 Text File

2804 A file that contains characters organized into one or more lines. The lines do not contain NUL
2805 characters and none can exceed `{LINE_MAX}` bytes in length, including the `<newline>`.
2806 Although IEEE Std 1003.1-200x does not distinguish between text files and binary files (see the
2807 ISO C standard), many utilities only produce predictable or meaningful output when operating
2808 on text files. The standard utilities that have such restrictions always specify *text files* in their

2809 STDIN or INPUT FILES sections. |

2810 **3.393 Thread** |

2811 A single flow of control within a process. Each thread has its own thread ID, scheduling priority |
2812 and policy, *errno* value, thread-specific key/value bindings, and the required system resources to |
2813 support a flow of control. Anything whose address may be determined by a thread, including |
2814 but not limited to static variables, storage obtained via *malloc()*, directly addressable storage |
2815 obtained through implementation-defined functions, and automatic variables, are accessible to |
2816 all threads in the same process.

2817 **Note:** The *malloc()* function is defined in detail in the System Interfaces volume of |
2818 IEEE Std 1003.1-200x.

2819 **3.394 Thread ID**

2820 Each thread in a process is uniquely identified during its lifetime by a value of type **pthread_t** |
2821 called a thread ID. |

2822 **3.395 Thread List** |

2823 An ordered set of runnable threads that all have the same ordinal value for their priority. |

2824 The ordering of threads on the list is determined by a scheduling policy or policies. The set of |
2825 thread lists includes all runnable threads in the system.

2826 **3.396 Thread-Safe**

2827 A function that may be safely invoked concurrently by multiple threads. Each function defined |
2828 in the System Interfaces volume of IEEE Std 1003.1-200x is thread-safe unless explicitly stated |
2829 otherwise. Examples are any “pure” function, a function which holds a mutex locked while it is |
2830 accessing static storage, or objects shared among threads. |

2831 **3.397 Thread-Specific Data Key** |

2832 A process global handle of type **pthread_key_t** which is used for naming thread-specific data. |

2833 Although the same key value may be used by different threads, the values bound to the key by |
2834 *pthread_setspecific()* and accessed by *pthread_getspecific()* are maintained on a per-thread basis |
2835 and persist for the life of the calling thread.

2836 **Note:** The *pthread_getspecific()* and *pthread_setspecific()* functions are defined in detail in the System |
2837 Interfaces volume of IEEE Std 1003.1-200x.

2838 **3.398 Tilde**

2839 The character '~'.

2840 3.399 Timeouts

2841 A method of limiting the length of time an interface will block; see also Section 3.76 (on page 43).

2842 3.400 Timer

2843 A mechanism that can notify a thread when the time as measured by a particular clock has
2844 reached or passed a specified value, or when a specified amount of time has passed.

2845 3.401 Timer Overrun

2846 A condition that occurs each time a timer, for which there is already an expiration signal queued
2847 to the process, expires.

2848 3.402 Token

2849 In the shell command language, a sequence of characters that the shell considers as a single unit
2850 when reading input. A token is either an operator or a word.

2851 **Note:** The rules for reading input are defined in detail in the Shell and Utilities volume of
2852 IEEE Std 1003.1-200x, Section 2.3, Token Recognition.

2853 3.403 Trace Analyzer Process

2854 A process that extracts trace events from a trace stream to retrieve information about the
2855 behavior of an application.

2856 3.404 Trace Controller Process

2857 A process that creates a trace stream for tracing a process.

2858 3.405 Trace Event

2859 A data object that represents an action executed by the system, and that is recorded in a trace
2860 stream.

2861 3.406 Trace Event Type

2862 A data object type that defines a class of trace event.

2863 3.407 Trace Event Type Mapping

2864 A one-to-one mapping between trace event types and trace event names.

2865 3.408 Trace Filter

2866 A filter that allows the trace controller process to specify those trace event types that are to be
2867 ignored; that is, not generated.

2868 3.409 Trace Generation Version

2869 A data object that is an implementation-defined character string, generated by the trace system
2870 and describing the origin and version of the trace system. |

2871 3.410 Trace Log |

2872 The flushed image of a trace stream, if the trace stream is created with a trace log. |

2873 3.411 Trace Point

2874 An action that may cause a trace event to be generated. |

2875 3.412 Trace Stream |

2876 An opaque object that contains trace events plus internal data needed to interpret those trace
2877 events. |

2878 3.413 Trace Stream Identifier

2879 A handle to manage tracing operations in a trace stream.

2880 3.414 Trace System

2881 A system that allows both system and user trace events to be generated into a trace stream.
2882 These trace events can be retrieved later. |

2883 3.415 Traced Process |

2884 A process for which at least one trace stream has been created. A traced process is also called a
2885 target process. |

2886 3.416 Tracing Status of a Trace Stream

2887 A status that describes the state of an active trace stream. The tracing status of a trace stream can
2888 be retrieved from the trace stream attributes. An active trace stream can be in one of two states:
2889 running or suspended.

2890 3.417 Typed Memory Name Space

2891 A system-wide name space that contains the names of the typed memory objects present in the
2892 system. It is configurable for a given implementation.

2893 3.418 Typed Memory Object

2894 A combination of a typed memory pool and a typed memory port. The entire contents of the
2895 pool are accessible from the port. The typed memory object is identified through a name that
2896 belongs to the typed memory name space.

2897 3.419 Typed Memory Pool

2898 An extent of memory with the same operational characteristics. Typed memory pools may be
2899 contained within each other.

2900 3.420 Typed Memory Port

2901 A hardware access path to one or more typed memory pools.

2902 3.421 Unbind

2903 Remove the association between a network address and an endpoint.

2904 3.422 Unit Data

2905 See *Datagram* in Section 3.123 (on page 49).

2906 3.423 Upshifting

2907 The conversion of a lowercase character that has a single-character uppercase representation
2908 into this uppercase representation. |

2909 3.424 User Database |

2910 A system database of implementation-defined format that contains at least the following |
2911 information for each user ID:

- 2912 • User name
- 2913 • Numerical user ID
- 2914 • Initial numerical group ID
- 2915 • Initial working directory
- 2916 • Initial user program

2917 The initial numerical group ID is used by the *newgrp* utility. Any other circumstances under
2918 which the initial values are operative are implementation-defined.

2919 If the initial user program field is null, an implementation-defined program is used.

2920 If the initial working directory field is null, the interpretation of that field is implementation-
2921 defined.

2922 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of IEEE Std 1003.1-200x.

2923 **3.425 User ID**

2924 A non-negative integer that is used to identify a system user. When the identity of a user is
2925 associated with a process, a user ID value is referred to as a real user ID, an effective user ID, or a
2926 saved set-user-ID.

2927 **3.426 User Name**

2928 A string that is used to identify a user; see also Section 3.424 (on page 90). To be portable across
2929 systems conforming to IEEE Std 1003.1-200x, the value is composed of characters from the
2930 portable filename character set. The hyphen should not be used as the first character of a
2931 portable user name.

2932 **3.427 User Trace Event**

2933 A trace event that is generated explicitly by the application as a result of a call to
2934 *posix_trace_event()*.

2935 **3.428 Utility**

2936 A program, excluding special built-in utilities provided as part of the Shell Command Language,
2937 that can be called by name from a shell to perform a specific task, or related set of tasks.

2938 **Note:** For further information on special built-in utilities, see the Shell and Utilities volume of
2939 IEEE Std 1003.1-200x, Section 2.14, Special Built-In Utilities.

2940 **3.429 Variable**

2941 In the shell command language, a named parameter.

2942 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.5,
2943 Parameters and Variables.

2944 **3.430 Vertical-Tab Character (<vertical-tab>)**

2945 A character that in the output stream indicates that printing should start at the next vertical
2946 tabulation position. It is the character designated by '`\v`' in the C language. If the current
2947 position is at or past the last defined vertical tabulation position, the behavior is unspecified. It is
2948 unspecified whether this character is the exact sequence transmitted to an output device by the
2949 system to accomplish the tabulation.

2950 **3.431 White Space** |

2951 A sequence of one or more characters that belong to the **space** character class as defined via the |
 2952 *LC_CTYPE* category in the current locale. |

2953 In the POSIX locale, white space consists of one or more <blank>s (<space>s and <tab>s), |
 2954 <newline>s, <carriage-return>s, <form-feed>s, and <vertical-tab>s. |

2955 **3.432 Wide-Character Code (C Language)** |

2956 An integer value corresponding to a single graphic symbol or control code. |

2957 **Note:** C Language Wide-Character Codes are defined in detail in Section 6.3 (on page 115). |

2958 **3.433 Wide-Character Input/Output Functions** |

2959 The functions that perform wide-oriented input from streams or wide-oriented output to |
 2960 streams: *fgetwc()*, *fputwc()*, *fputws()*, *fwprintf()*, *fwscanf()*, *getwc()*, *getwchar()*, *getws()*, *putwc()*, |
 2961 *putwchar()*, *ungetwc()*, *vfwprintf()*, *vwprintf()*, *wprintf()*, and *wscanf()*. |

2962 **Note:** These functions are defined in detail in the System Interfaces volume of IEEE Std 1003.1-200x. |

2963 **3.434 Wide-Character String** |

2964 A contiguous sequence of wide-character codes terminated by and including the first null wide- |
 2965 character code. |

2966 **3.435 Word** |

2967 In the shell command language, a token other than an operator. In some cases a word is also a |
 2968 portion of a word token: in the various forms of parameter expansion, such as $\${name-word}$, and |
 2969 variable assignment, such as *name=word*, the word is the portion of the token depicted by *word*. |
 2970 The concept of a word is no longer applicable following word expansions—only fields remain. |

2971 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section |
 2972 2.6.2, Parameter Expansion and the Shell and Utilities volume of IEEE Std 1003.1-200x, Section |
 2973 2.6, Word Expansions. |

2974 **3.436 Working Directory (or Current Working Directory)** |

2975 A directory, associated with a process, that is used in pathname resolution for pathnames that |
 2976 do not begin with a slash. |

2977 **3.437 Worldwide Portability Interface** |

2978 Functions for handling characters in a codeset-independent manner. |

2979 **3.438 Write**

2980 To output characters to a file, such as standard output or standard error. Unless otherwise
2981 stated, standard output is the default output destination for all uses of the term write; see the
2982 distinction between display and write in Section 3.132 (on page 50).

2983 **3.439 XSI**

2984 The X/Open System Interface is the core application programming interface for C and *sh*
2985 programming for systems conforming to the Single UNIX Specification. This is a superset of the
2986 mandatory requirements for conformance to IEEE Std 1003.1-200x. |

2987 **3.440 XSI-Conformant** |

2988 A system which allows an application to be built using a set of services that are consistent across |
2989 all systems that conform to IEEE Std 1003.1-200x and that support the XSI extension. |

2990 **Note:** See also Chapter 2 (on page 15).

2991 **3.441 Zombie Process**

2992 A process that has terminated and that is deleted when its exit status has been reported to
2993 another process which is waiting for that process to terminate.

2994 **3.442 ± 0**

2995 The algebraic sign provides additional information about any variable that has the value zero
2996 when the representation allows the sign to be determined. |

General Concepts

2998

2999 For the purposes of IEEE Std 1003.1-200x, the general concepts given in Chapter 4 apply.

3000 **Note:** No shading to denote extensions or options occurs in this chapter. Where the terms and
 3001 definitions given in this chapter are used elsewhere in text related to extensions and options,
 3002 they are shaded as appropriate.

3003 4.1 Concurrent Execution

3004 Functions that suspend the execution of the calling thread shall not cause the execution of other
 3005 threads to be indefinitely suspended.

3006 4.2 Directory Protection

3007 If a directory is writable and the mode bit `S_ISVTX` is set on the directory, a process may remove
 3008 or rename files within that directory only if one or more of the following is true:

- 3009 • The effective user ID of the process is the same as that of the owner ID of the file.
- 3010 • The effective user ID of the process is the same as that of the owner ID of the directory.
- 3011 • The process has appropriate privileges.

3012 If the `S_ISVTX` bit is set on a non-directory file, the behavior is unspecified.

3013 4.3 Extended Security Controls

3014 An implementation may provide implementation-defined extended security controls (see
 3015 Section 3.159 (on page 54)). These permit an implementation to provide security mechanisms to
 3016 implement different security policies than those described in IEEE Std 1003.1-200x. These
 3017 mechanisms shall not alter or override the defined semantics of any of the interfaces in
 3018 IEEE Std 1003.1-200x.

3019 4.4 File Access Permissions

3020 The standard file access control mechanism uses the file permission bits, as described below.

3021 Implementations may provide *additional* or *alternate* file access control mechanisms, or both. An
 3022 additional access control mechanism shall only further restrict the access permissions defined by
 3023 the file permission bits. An alternate file access control mechanism shall:

- 3024 • Specify file permission bits for the file owner class, file group class, and file other class of that
 3025 file, corresponding to the access permissions.
- 3026 • Be enabled only by explicit user action, on a per-file basis by the file owner or a user with the
 3027 appropriate privilege.
- 3028 • Be disabled for a file after the file permission bits are changed for that file with `chmod()`. The
 3029 disabling of the alternate mechanism need not disable any additional mechanisms supported

3030 by an implementation.

3031 Whenever a process requests file access permission for read, write, or execute/search, if no
3032 additional mechanism denies access, access shall be determined as follows: |

3033 • If a process has the appropriate privilege:

3034 — If read, write, or directory search permission is requested, access shall be granted. |

3035 — If execute permission is requested, access shall be granted if execute permission is |
3036 granted to at least one user by the file permission bits or by an alternate access control |
3037 mechanism; otherwise, access shall be denied. |

3038 • Otherwise:

3039 — The file permission bits of a file contain read, write, and execute/search permissions for
3040 the file owner class, file group class, and file other class.

3041 — Access shall be granted if an alternate access control mechanism is not enabled and the |
3042 requested access permission bit is set for the class (file owner class, file group class, or file |
3043 other class) to which the process belongs, or if an alternate access control mechanism is |
3044 enabled and it allows the requested access; otherwise, access shall be denied. |

3045 4.5 File Hierarchy

3046 Files in the system are organized in a hierarchical structure in which all of the non-terminal
3047 nodes are directories and all of the terminal nodes are any other type of file. Since multiple |
3048 directory entries may refer to the same file, the hierarchy is properly described as a *directed* |
3049 *graph*.

3050 4.6 Filenames

3051 For a filename to be portable across implementations conforming to IEEE Std 1003.1-200x, it |
3052 shall consist only of the portable filename character set as defined in Section 3.276 (on page 70). |

3053 The hyphen character shall not be used as the first character of a portable filename. Uppercase
3054 and lowercase letters shall retain their unique identities between conforming implementations.
3055 In the case of a portable pathname, the slash character may also be used.

3056 4.7 File Times Update

3057 Each file has three distinct associated time values: *st_atime*, *st_mtime*, and *st_ctime*. The *st_atime*
3058 field is associated with the times that the file data is accessed; *st_mtime* is associated with the
3059 times that the file data is modified; and *st_ctime* is associated with the times that the file status is
3060 changed. These values are returned in the file characteristics structure, as described in
3061 `<sys/stat.h>`.

3062 Each function or utility in IEEE Std 1003.1-200x that reads or writes data or changes file status
3063 indicates which of the appropriate time-related fields shall be “marked for update”. If an
3064 implementation of such a function or utility marks for update a time-related field not specified
3065 by IEEE Std 1003.1-200x, this shall be documented, except that any changes caused by pathname
3066 resolution need not be documented. For the other functions or utilities in IEEE Std 1003.1-200x
3067 (those that are not explicitly required to read or write file data or change file status, but that in
3068 some implementations happen to do so), the effect is unspecified.

3069 An implementation may update fields that are marked for update immediately, or it may update
3070 such fields periodically. At an update point in time, any marked fields shall be set to the current
3071 time and the update marks shall be cleared. All fields that are marked for update shall be
3072 updated when the file ceases to be open by any process, or when a *stat()*, *fstat()*, or *lstat()* is
3073 performed on the file. Other times at which updates are done are unspecified. Marks for update,
3074 and updates themselves, are not done for files on read-only file systems; see Section 3.304 (on
3075 page 74).

3076 **4.8 Host and Network Byte Orders**

3077 When data is transmitted over the network, it is sent as a sequence of octets (8-bit unsigned
3078 values). If an entity (such as an address or a port number) can be larger than 8 bits, it needs to be
3079 stored in several octets. The convention is that all such values are stored with 8 bits in each octet,
3080 and with the first (lowest-addressed) octet holding the most-significant bits. This is called
3081 “network byte order”.

3082 Network byte order may not be convenient for processing actual values. For this, it is more
3083 sensible for values to be stored as ordinary integers. This is known as “host byte order”. In host
3084 byte order:

- 3085 • The most significant bit might not be stored in the first byte in address order.
- 3086 • Bits might not be allocated to bytes in any obvious order at all.

3087 8-bit values stored in **uint8_t** objects do not require conversion to or from host byte order, as
3088 they have the same representation. 16 and 32-bit values can be converted using the *htonl()*,
3089 *htons()*, *ntohl()*, and *ntohs()* functions. When reading data that is to be converted to host byte
3090 order, it should either be received directly into a **uint16_t** or **uint32_t** object or should be copied
3091 from an array of bytes using *memcpy()* or similar. Passing the data through other types could
3092 cause the byte order to be changed. Similar considerations apply when sending data.

3093 **4.9 Measurement of Execution Time**

3094 The mechanism used to measure execution time shall be implementation-defined. The
3095 implementation shall also define to whom the CPU time that is consumed by interrupt handlers
3096 and system services on behalf of the operating system will be charged. See Section 3.117 (on
3097 page 49).

3098 4.10 Memory Synchronization

3099 Applications shall ensure that access to any memory location by more than one thread of control
 3100 (threads or processes) is restricted such that no thread of control can read or modify a memory
 3101 location while another thread of control may be modifying it. Such access is restricted using
 3102 functions that synchronize thread execution and also synchronize memory with respect to other
 3103 threads. The following functions synchronize memory with respect to other threads:

3104	<i>fork()</i>	<i>pthread_mutex_timedlock()</i>	<i>pthread_rwlock_tryrdlock()</i>
3105	<i>pthread_barrier_wait()</i>	<i>pthread_mutex_trylock()</i>	<i>pthread_rwlock_trywrlock()</i>
3106	<i>pthread_cond_broadcast()</i>	<i>pthread_mutex_unlock()</i>	<i>pthread_rwlock_unlock()</i>
3107	<i>pthread_cond_signal()</i>	<i>pthread_spin_lock()</i>	<i>pthread_rwlock_wrlock()</i>
3108	<i>pthread_cond_timedwait()</i>	<i>pthread_spin_trylock()</i>	<i>sem_post()</i>
3109	<i>pthread_cond_wait()</i>	<i>pthread_spin_unlock()</i>	<i>sem_trywait()</i>
3110	<i>pthread_create()</i>	<i>pthread_rwlock_rdlock()</i>	<i>sem_wait()</i>
3111	<i>pthread_join()</i>	<i>pthread_rwlock_timedrdlock()</i>	<i>wait()</i>
3112	<i>pthread_mutex_lock()</i>	<i>pthread_rwlock_timedwrlock()</i>	<i>waitpid()</i>

3113 Unless explicitly stated otherwise, if one of the above functions returns an error, it is unspecified
 3114 whether the invocation causes memory to be synchronized.

3115 Applications may allow more than one thread of control to read a memory location
 3116 simultaneously.

3117 4.11 Pathname Resolution

3118 Pathname resolution is performed for a process to resolve a pathname to a particular file in a file
 3119 hierarchy. There may be multiple pathnames that resolve to the same file.

3120 Each filename in the pathname is located in the directory specified by its predecessor (for
 3121 example, in the pathname fragment **a/b**, file **b** is located in directory **a**). Pathname resolution
 3122 shall fail if this cannot be accomplished. If the pathname begins with a slash, the predecessor of
 3123 the first filename in the pathname shall be taken to be the root directory of the process (such
 3124 pathnames are referred to as *absolute pathnames*). If the pathname does not begin with a slash, the
 3125 predecessor of the first filename of the pathname shall be taken to be the current working
 3126 directory of the process (such pathnames are referred to as *relative pathnames*).

3127 The interpretation of a pathname component is dependent on the value of {NAME_MAX} and
 3128 _POSIX_NO_TRUNC associated with the path prefix of that component. If any pathname
 3129 component is longer than {NAME_MAX}, the implementation shall consider this an error.

3130 A pathname that contains at least one non-slash character and that ends with one or more
 3131 trailing slashes shall be resolved as if a single dot character ('.') were appended to the
 3132 pathname.

3133 If a symbolic link is encountered during pathname resolution, the behavior shall depend on
 3134 whether the pathname component is at the end of the pathname and on the function being
 3135 performed. If all of the following are true, then pathname resolution is complete:

- 3136 1. This is the last pathname component of the pathname.
- 3137 2. The pathname has no trailing slash.
- 3138 3. The function is required to act on the symbolic link itself, or certain arguments direct that
 3139 the function act on the symbolic link itself.

3140 In all other cases, the system shall prefix the remaining pathname, if any, with the contents of the
 3141 symbolic link. If the combined length exceeds {PATH_MAX}, and the implementation considers
 3142 this to be an error, *errno* shall be set to [ENAMETOOLONG] and an error indication shall be
 3143 returned. Otherwise, the resolved pathname shall be the resolution of the pathname just created.
 3144 If the resulting pathname does not begin with a slash, the predecessor of the first filename of the
 3145 pathname is taken to be the directory containing the symbolic link.

3146 If the system detects a loop in the pathname resolution process, it shall set *errno* to [ELOOP] and
 3147 return an error indication. The same may happen if during the resolution process more symbolic
 3148 links were followed than the implementation allows. This implementation-defined limit shall
 3149 not be smaller than {SYMLOOP_MAX}.

3150 The special filename dot shall refer to the directory specified by its predecessor. The special |
 3151 filename dot-dot shall refer to the parent directory of its predecessor directory. As a special case, |
 3152 in the root directory, dot-dot may refer to the root directory itself.

3153 A pathname consisting of a single slash shall resolve to the root directory of the process. A null |
 3154 pathname shall not be successfully resolved. A pathname that begins with two successive |
 3155 slashes may be interpreted in an implementation-defined manner, although more than two |
 3156 leading slashes shall be treated as a single slash.

3157 4.12 Process ID Reuse

3158 A process group ID shall not be reused by the system until the process group lifetime ends.

3159 A process ID shall not be reused by the system until the process lifetime ends. In addition, if
 3160 there exists a process group whose process group ID is equal to that process ID, the process ID
 3161 shall not be reused by the system until the process group lifetime ends. A process that is not a
 3162 system process shall not have a process ID of 1.

3163 4.13 Scheduling Policy

3164 A scheduling policy affects process or thread ordering:

- 3165 • When a process or thread is a running thread and it becomes a blocked thread
- 3166 • When a process or thread is a running thread and it becomes a preempted thread
- 3167 • When a process or thread is a blocked thread and it becomes a runnable thread
- 3168 • When a running thread calls a function that can change the priority or scheduling policy of a
 3169 process or thread
- 3170 • In other scheduling policy-defined circumstances

3171 Conforming implementations shall define the manner in which each of the scheduling policies |
 3172 may modify the priorities or otherwise affect the ordering of processes or threads at each of the |
 3173 occurrences listed above. Additionally, conforming implementations shall define in what other |
 3174 circumstances and in what manner each scheduling policy may modify the priorities or affect |
 3175 the ordering of processes or threads.

3176 4.14 Seconds Since the Epoch

3177 A value that approximates the number of seconds that have elapsed since the Epoch. A
 3178 Coordinated Universal Time name (specified in terms of seconds (*tm_sec*), minutes (*tm_min*),
 3179 hours (*tm_hour*), days since January 1 of the year (*tm_yday*), and calendar year minus 1900
 3180 (*tm_year*)) is related to a time represented as seconds since the Epoch, according to the
 3181 expression below.

3182 If the year is <1970 or the value is negative, the relationship is undefined. If the year is ≥1970 and
 3183 the value is non-negative, the value is related to a Coordinated Universal Time name according
 3184 to the C-language expression, where *tm_sec*, *tm_min*, *tm_hour*, *tm_yday*, and *tm_year* are all
 3185 integer types:

$$\begin{aligned}
 &tm_sec + tm_min*60 + tm_hour*3600 + tm_yday*86400 + \\
 & (tm_year-70)*31536000 + ((tm_year-69)/4)*86400 - \\
 & ((tm_year-1)/100)*86400 + ((tm_year+299)/400)*86400
 \end{aligned}$$

3189 The relationship between the actual time of day and the current value for seconds since the
 3190 Epoch is unspecified.

3191 How any changes to the value of seconds since the Epoch are made to align to a desired
 3192 relationship with the current actual time are made is implementation-defined. As represented in
 3193 seconds since the Epoch, each and every day shall be accounted for by exactly 86 400 seconds.

3194 **Note:** The last three terms of the expression add in a day for each year that follows a leap year
 3195 starting with the first leap year since the Epoch. The first term adds a day every 4 years |
 3196 starting in 1973, the second subtracts a day back out every 100 years starting in 2001, and the |
 3197 third adds a day back in every 400 years starting in 2001. The divisions in the formula are |
 3198 integer divisions; that is, the remainder is discarded leaving only the integer quotient. |

3199 4.15 Semaphore

3200 A minimum synchronization primitive to serve as a basis for more complex synchronization
 3201 mechanisms to be defined by the application program.

3202 For the semaphores associated with the Semaphores option, a semaphore is represented as a
 3203 shareable resource that has a non-negative integer value. When the value is zero, there is a
 3204 (possibly empty) set of threads awaiting the availability of the semaphore.

3205 For the semaphores associated with the X/Open System Interface Extension (XSI), a semaphore
 3206 is a positive integer (0 through 32767). The *semget()* function can be called to create a set or array
 3207 of semaphores. A semaphore set can contain one or more semaphores up to an implementation-
 3208 defined value.

3209 Semaphore Lock Operation

3210 An operation that is applied to a semaphore. If, prior to the operation, the value of the
 3211 semaphore is zero, the semaphore lock operation shall cause the calling thread to be blocked and
 3212 added to the set of threads awaiting the semaphore; otherwise, the value shall be decremented. |

3213 **Semaphore Unlock Operation**

3214 An operation that is applied to a semaphore. If, prior to the operation, there are any threads in
 3215 the set of threads awaiting the semaphore, then some thread from that set shall be removed from
 3216 the set and becomes unblocked; otherwise, the semaphore value shall be incremented. |

3217 **4.16 Thread-Safety**

3218 Refer to the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.9, Threads.

3219 **4.17 Tracing**

3220 The trace system allows a traced process to have a selection of events created for it. Traces
 3221 consist of streams of trace event types.

3222 A trace event type is identified on the one hand by a trace event type name, also referenced as a
 3223 trace event name, and on the other hand by a trace event type identifier. A trace event name is a
 3224 human-readable string. A trace event type identifier is an opaque identifier used by the trace |
 3225 system. There shall be a one-to-one relationship between trace event type identifiers and trace |
 3226 event names for a given trace stream and also for a given traced process. The trace event type |
 3227 identifier shall be generated automatically from a trace event name by the trace system either |
 3228 when a trace controller process invokes *posix_trace_trid_eventid_open()* or when an instrumented |
 3229 application process invokes *posix_trace_eventid_open()*. Trace event type identifiers are used to |
 3230 filter trace event types, to allow interpretation of user data, and to identify the kind of trace point
 3231 that generated a trace event.

3232 Each trace event shall be of a particular trace event type, and associated with a trace event type |
 3233 identifier. The execution of a trace point shall generate a trace event if a trace stream has been |
 3234 created and started for the process that executed the trace point and if the corresponding trace |
 3235 event type identifier is not ignored by filtering. |

3236 A generated trace event shall be recorded in a trace stream, and optionally also in a trace log if a
 3237 trace log is associated with the trace stream, except that:

- 3238 • For a trace stream, if no resources are available for the event, the event is lost.
- 3239 • For a trace log, if no resources are available for the event, or a flush operation does not
 3240 succeed, the event is lost.

3241 A trace event recorded in an active trace stream may be retrieved by an application having the
 3242 appropriate privileges.

3243 A trace event recorded in a trace log may be retrieved by an application having the appropriate
 3244 privileges after opening the trace log as a pre-recorded trace stream, with the function
 3245 *posix_trace_open()*.

3246 When a trace event is reported it is possible to retrieve the following:

- 3247 • A trace event type identifier
- 3248 • A timestamp
- 3249 • The process ID of the traced process, if the trace event is process-dependent
- 3250 • Any optional trace event data including its length

- 3251 • If the Threads option is supported, the thread ID, if the trace event is process-dependent
 3252 • The program address at which the trace point was invoked
- 3253 Trace events may be mapped from trace event types to trace event names. One such mapping |
 3254 shall be associated with each trace stream. An active trace stream is associated with a traced |
 3255 process, and also with its children if the Trace Inherit option is supported and also the |
 3256 inheritance policy is set to `_POSIX_TRACE_INHERIT`. Therefore each traced process has a |
 3257 mapping of the trace event names to trace event type identifiers that have been defined for that |
 3258 process. |
- 3259 Traces can be recorded into either trace streams or trace logs. |
- 3260 The implementation and format of a trace stream are unspecified. A trace stream need not be |
 3261 and generally is not persistent. A trace stream may be either active or pre-recorded:
- 3262 • An active trace stream is a trace stream that has been created and has not yet been shut |
 3263 down. It can be of one of the two following classes:
- 3264 1. An active trace stream without a trace log that was created with the *posix_trace_create()* |
 3265 function
- 3266 2. If the Trace Log option is supported, an active trace stream with a trace log that was |
 3267 created with the *posix_trace_create_withlog()* function
- 3268 • A pre-recorded trace stream is a trace stream that was opened from a trace log object using |
 3269 the *posix_trace_open()* function.
- 3270 An active trace stream can loop. This behavior means that when the resources allocated by the |
 3271 trace system for the trace stream are exhausted, the trace system reuses the resources associated |
 3272 with the oldest recorded trace events to record new trace events.
- 3273 If the Trace Log option is supported, an active trace stream with a trace log can be flushed. This |
 3274 operation causes the trace system to write trace events from the trace stream to the associated |
 3275 trace log, following the defined policies or using an explicit function call. After this operation, |
 3276 the trace system may reuse the resources associated with the flushed trace events.
- 3277 An active trace stream with or without a trace log can be cleared. This operation shall cause all |
 3278 the resources associated with this trace stream to be reinitialized. The trace stream shall behave |
 3279 as if it was returning from its creation, except that the mapping of trace event type identifiers to |
 3280 trace event names shall not be cleared. If a trace log was associated with this trace stream, the |
 3281 trace log shall also be reinitialized. |
- 3282 A trace log shall be recorded when the *posix_trace_shutdown()* operation is invoked or during |
 3283 tracing, depending on the tracing strategy which is defined by a log policy. After the trace |
 3284 stream has been shut down, the trace information can be retrieved from the associated trace log |
 3285 using the same interface used to retrieve information from an active trace stream.
- 3286 For a traced process, if the Trace Inherit option is supported and the trace stream's inheritance |
 3287 attribute is `_POSIX_TRACE_INHERIT`, the initial targeted traced process shall be traced together |
 3288 with all of its future children. The *posix_pid* member of each trace event in a trace stream shall be |
 3289 the process ID of the traced process. |
- 3290 Each trace point may be an implementation-defined action such as a context switch, or an |
 3291 application-programmed action such as a call to a specific operating system service (for |
 3292 example, *fork()*) or a call to *posix_trace_event()*.
- 3293 Trace points may be filtered. The operation of the filter is to filter out (ignore) selected trace |
 3294 events. By default, no trace events are filtered.

3295 The results of the tracing operations can be analyzed and monitored by a trace controller process
3296 or a trace analyzer process.

3297 Only the trace controller process has control of the trace stream it has created. The control of the
3298 operation of a trace stream is done using its corresponding trace stream identifier. The trace
3299 controller process is able to:

- 3300 • Initialize the attributes of a trace stream
- 3301 • Create the trace stream
- 3302 • Start and stop tracing
- 3303 • Know the mapping of the traced process
- 3304 • If the Trace Event Filter option is supported, filter the type of trace events to be recorded
- 3305 • Shut the trace stream down

3306 A traced process may also be a trace controller process. Only the trace controller process can |
3307 control its trace stream(s). A trace stream created by a trace controller process shall be shut |
3308 down if its controller process terminates or executes another file. |

3309 A trace controller process may also be a trace analyzer process. Trace analysis can be done
3310 concurrently with the traced process or can be done off-line, in the same or in a different
3311 platform.

3312 **4.18 Treatment of Error Conditions for Mathematical Functions**

3313 For all the functions in the `<math.h>` header, an application wishing to check for error situations
3314 should set `errno` to 0 and call `feclearexcept(FE_ALL_EXCEPT)` before calling the function. On
3315 return, if `errno` is non-zero or `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW |`
3316 `FE_UNDERFLOW)` is non-zero, an error has occurred.

3317 The following error conditions are defined for all functions in the `<math.h>` header.

3318 **4.18.1 Domain Error**

3319 A *domain error* shall occur if an input argument is outside the domain over which the
3320 mathematical function is defined. The description of each function lists any required domain
3321 errors; an implementation may define additional domain errors, provided that such errors are
3322 consistent with the mathematical definition of the function.

3323 On a domain error, the function shall return an implementation-defined value; if the integer |
3324 expression (`math_errhandling & MATH_ERRNO`) is non-zero, `errno` shall be set to [EDOM]; if |
3325 the integer expression (`math_errhandling & MATH_ERREXCEPT`) is non-zero, the “invalid” |
3326 floating-point exception shall be raised. |

3327 **4.18.2 Pole Error**

3328 A *pole error* occurs if the mathematical result of the function is an exact infinity (for example,
3329 $\log(0.0)$).

3330 On a pole error, the function shall return the value of the macro HUGE_VAL, HUGE_VALF, or
3331 HUGE_VALL according to the return type, with the same sign as the correct value of the
3332 function; if the integer expression (math_errhandling & MATH_ERRNO) is non-zero, *errno* shall
3333 be set to [ERANGE]; if the integer expression (math_errhandling & MATH_ERREXCEPT) is
3334 non-zero, the “divide-by-zero” floating-point exception shall be raised.

3335 **4.18.3 Range Error**

3336 A *range error* shall occur if the finite mathematical result of the function cannot be represented in
3337 an object of the specified type, due to extreme magnitude.

3338 **4.18.3.1 Result Overflows**

3339 A floating result overflows if the magnitude of the mathematical result is finite but so large that
3340 the mathematical result cannot be represented without extraordinary roundoff error in an object
3341 of the specified type. If a floating result overflows and default rounding is in effect, then the
3342 function shall return the value of the macro HUGE_VAL, HUGE_VALF, or HUGE_VALL
3343 according to the return type, with the same sign as the correct value of the function; if the integer
3344 expression (math_errhandling & MATH_ERRNO) is non-zero, *errno* shall be set to [ERANGE]; if
3345 the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, the “overflow”
3346 floating-point exception shall be raised.

3347 **4.18.3.2 Result Underflows**

3348 The result underflows if the magnitude of the mathematical result is so small that the
3349 mathematical result cannot be represented, without extraordinary roundoff error, in an object of
3350 the specified type. If the result underflows, the function shall return an implementation-defined
3351 value whose magnitude is no greater than the smallest normalized positive number in the
3352 specified type; if the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
3353 whether *errno* is set to [ERANGE] is implementation-defined; if the integer expression
3354 (math_errhandling & MATH_ERREXCEPT) is non-zero, whether the “underflow” floating-point
3355 exception is raised is implementation-defined.

3356 **4.19 Treatment of NaN Arguments for the Mathematical Functions**

3357 For functions called with a NaN argument, no errors shall occur and a NaN shall be returned,
3358 except where stated otherwise.

3359 If a function with one or more NaN arguments returns a NaN result, the result should be the
3360 same as one of the NaN arguments (after possible type conversion), except perhaps for the sign.

3361 On implementations that support the IEC 60559:1989 standard floating point, functions with
3362 signaling NaN argument(s) shall be treated as if the function were called with an argument that
3363 is a required domain error and shall return a quiet NaN result, except where stated otherwise.

3364 **Note:** The function might never see the signaling NaN, since it might trigger when the arguments are
3365 evaluated during the function call.

3366 On implementations that support the IEC 60559:1989 standard floating point, for those
3367 functions that do not have a documented domain error, the following shall apply:

3368 These functions shall fail if:
 3369 Domain Error Any argument is a signaling NaN.
 3370 Either, the integer expression (`math_errhandling & MATH_ERRNO`) is non-zero and *errno*
 3371 shall be set to [EDOM], or the integer expression (`math_errhandling & MATH_ERREXCEPT`)
 3372 is non-zero and the invalid floating-point exception shall be raised.

3373 4.20 Utility

3374 A utility program shall be either an executable file, such as might be produced by a compiler or
 3375 linker system from computer source code, or a file of shell source code, directly interpreted by
 3376 the shell. The program may have been produced by the user, provided by the system
 3377 implementor, or acquired from an independent distributor.

3378 The system may implement certain utilities as shell functions (see the Shell and Utilities volume
 3379 of IEEE Std 1003.1-200x, Section 2.9.5, Function Definition Command) or built-in utilities, but
 3380 only an application that is aware of the command search order described in the Shell and
 3381 Utilities volume of IEEE Std 1003.1-200x, Section 2.9.1.1, Command Search and Execution or of
 3382 performance characteristics can discern differences between the behavior of such a function or
 3383 built-in utility and that of an executable file.

3384 4.21 Variable Assignment

3385 In the shell command language, a word consisting of the following parts:

3386 `varname=value`

3387 When used in a context where assignment is defined to occur and at no other time, the *value*
 3388 (representing a word or field) shall be assigned as the value of the variable denoted by *varname*.

3389 **Note:** For further information, see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section
 3390 2.9.1, Simple Commands.

3391 The *varname* and *value* parts shall meet the requirements for a name and a word, respectively, |
 3392 except that they are delimited by the embedded unquoted equals-sign, in addition to other |
 3393 delimiters. |

3394 **Note:** Additional delimiters are described in the Shell and Utilities volume of IEEE Std 1003.1-200x,
 3395 Section 2.3, Token Recognition.

3396 When a variable assignment is done, the variable shall be created if it did not already exist. If
 3397 *value* is not specified, the variable shall be given a null value.

3398 **Note:** An alternative form of variable assignment:

3399 `symbol=value`

3400 (where *symbol* is a valid word delimited by an equals-sign, but not a valid name) produces
 3401 unspecified results. The form `symbol=value` is used by the KornShell `name[expression]=value`
 3402 syntax.

File Format Notation

3404

3405 The STDIN, STDOUT, STDERR, INPUT FILES, and OUTPUT FILES sections of the utility
 3406 descriptions use a syntax to describe the data organization within the files, when that
 3407 organization is not otherwise obvious. The syntax is similar to that used by the System Interfaces
 3408 volume of IEEE Std 1003.1-200x *printf()* function, as described in this chapter. When used in
 3409 STDIN or INPUT FILES sections of the utility descriptions, this syntax describes the format that
 3410 could have been used to write the text to be read, not a format that could be used by the System
 3411 Interfaces volume of IEEE Std 1003.1-200x *scanf()* function to read the input file.

3412 The description of an individual record is as follows:

3413 "*format*", [*arg1*, *arg2*, . . . , *argn*]

3414 The *format* is a character string that contains three types of objects defined below:

- 3415 1. *Characters* that are not *escape sequences* or *conversion specifications*, as described below, shall
 3416 be copied to the output.
- 3417 2. *Escape Sequences* represent non-graphic characters.
- 3418 3. *Conversion Specifications* specify the output format of each argument; (see below).

3419 The following characters have the following special meaning in the format string:

3420 ' ' (An empty character position.) Represents one or more <blank>s.

3421 Δ Represents exactly one <space>.

3422 Table 5-1 lists escape sequences and associated actions on display devices capable of the action.

3423

Table 5-1 Escape Sequences and Associated Actions

3424

3425

3426

3427

3428

3429

3430

3431

3432

3433

3434

3435

3436

3437

3438

3439

Escape Sequence	Represents Character	Terminal Action
'\\'	backslash	Print the character '\\'.
'\a'	alert	Attempt to alert the user through audible or visible notification.
'\b'	backspace	Move the printing position to one column before the current position, unless the current position is the start of a line.
'\f'	form-feed	Move the printing position to the initial printing position of the next logical page.
'\n'	newline	Move the printing position to the start of the next line.
'\r'	carriage-return	Move the printing position to the start of the current line.
'\t'	tab	Move the printing position to the next tab position on the current line. If there are no more tab positions remaining on the line, the behavior is undefined.
'\v'	vertical-tab	Move the printing position to the start of the next vertical tab position. If there are no more vertical tab positions left on the page, the behavior is undefined.

3440

3441

Each conversion specification is introduced by the percent-sign character ('%'). After the character '%', the following shall appear in sequence: |

3442

3443

flags Zero or more *flags*, in any order, that modify the meaning of the conversion specification. |

3444

3445

3446

3447

field width An optional string of decimal digits to specify a minimum *field width*. For an output field, if the converted value has fewer bytes than the field width, it shall be padded on the left (or right, if the left-adjustment flag ('-'), described below, has been given) to the field width. |

3448

3449

3450

3451

3452

3453

3454

precision Gives the minimum number of digits to appear for the *d*, *o*, *i*, *u*, *x*, or *X* conversion specifiers (the field is padded with leading zeros), the number of digits to appear after the radix character for the *e* and *f* conversion specifiers, the maximum number of significant digits for the *g* conversion specifier; or the maximum number of bytes to be written from a string in the *s* conversion specifier. The precision shall take the form of a period ('.') followed by a decimal digit string; a null digit string is treated as zero. |

3455

3456

3457

conversion specifier characters

A conversion specifier character (see below) that indicates the type of conversion to be applied. |

3458

The *flag* characters and their meanings are:

3459

3460

3461

3462

3463

3464

3465

3466

3467

- The result of the conversion shall be left-justified within the field. |

+ The result of a signed conversion shall always begin with a sign ('+' or '-'). |

<space> If the first character of a signed conversion is not a sign, a <space> shall be prefixed to the result. This means that if the <space> and '+' flags both appear, the <space> flag shall be ignored. |

The value shall be converted to an alternative form. For *c*, *d*, *i*, *u*, and *s* conversion specifiers, the behavior is undefined. For the *o* conversion specifier, it shall increase the precision to force the first digit of the result to be a zero. For *x* or *X* conversion specifiers, a non-zero result has 0x or 0X prefixed to it, respectively. For |

3468		e, E, f, g, and G conversion specifiers, the result shall always contain a radix character, even if no digits follow the radix character. For g and G conversion specifiers, trailing zeros shall not be removed from the result as they usually are.
3469		
3470		
3471	0	For d, i, o, u, x, X, e, E, f, g, and G conversion specifiers, leading zeros (following any indication of sign or base) shall be used to pad to the field width; no space padding is performed. If the '0' and '-' flags both appear, the '0' flag shall be ignored. For d, i, o, u, x, and X conversion specifiers, if a precision is specified, the '0' flag shall be ignored. For other conversion specifiers, the behavior is undefined.
3472		
3473		
3474		
3475		
3476		
3477		Each conversion specifier character shall result in fetching zero or more arguments. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments shall be ignored.
3478		
3479		
3480		The <i>conversion specifiers</i> and their meanings are:
3481	d,i,o,u,x,X	The integer argument shall be written as signed decimal (d or i), unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal notation (x and X). The d and i specifiers shall convert to signed decimal in the style "[-]dddd". The x conversion specifier shall use the numbers and letters "0123456789abcdef" and the X conversion specifier shall use the numbers and letters "0123456789ABCDEF". The <i>precision</i> component of the argument shall specify the minimum number of digits to appear. If the value being converted can be represented in fewer digits than the specified minimum, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting a zero value with a precision of 0 shall be no characters. If both the field width and precision are omitted, the implementation may precede, follow, or precede and follow numeric arguments of types d, i, and u with <blank>s; arguments of type o (octal) may be preceded with leading zeros.
3482		
3483		
3484		
3485		
3486		
3487		
3488		
3489		
3490		
3491		
3492		
3493		
3494	f	The floating-point number argument shall be written in decimal notation in the style [-]ddd.ddd, where the number of digits after the radix character (shown here as a decimal point) shall be equal to the <i>precision</i> specification. The <i>LC_NUMERIC</i> locale category shall determine the radix character to use in this format. If the <i>precision</i> is omitted from the argument, six digits shall be written after the radix character; if the <i>precision</i> is explicitly 0, no radix character shall appear.
3495		
3496		
3497		
3498		
3499		
3500	e,E	The floating-point number argument shall be written in the style [-]d.ddde±dd (the symbol '±' indicates either a plus or minus sign), where there is one digit before the radix character (shown here as a decimal point) and the number of digits after it is equal to the precision. The <i>LC_NUMERIC</i> locale category shall determine the radix character to use in this format. When the precision is missing, six digits shall be written after the radix character; if the precision is 0, no radix character shall appear. The E conversion specifier shall produce a number with E instead of e introducing the exponent. The exponent shall always contain at least two digits. However, if the value to be written requires an exponent greater than two digits, additional exponent digits shall be written as necessary.
3501		
3502		
3503		
3504		
3505		
3506		
3507		
3508		
3509		
3510	g,G	The floating-point number argument shall be written in style f or e (or in style F or E in the case of a G conversion specifier), with the precision specifying the number of significant digits. The style used depends on the value converted: style e (or E) shall be used only if the exponent resulting from the conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the result. A radix character shall appear only if it is followed by a digit.
3511		
3512		
3513		
3514		
3515		

3516 c The integer argument shall be converted to an **unsigned char** and the resulting
 3517 byte shall be written.

3518 s The argument shall be taken to be a string and bytes from the string shall be
 3519 written until the end of the string or the number of bytes indicated by the *precision*
 3520 specification of the argument is reached. If the precision is omitted from the
 3521 argument, it shall be taken to be infinite, so all bytes up to the end of the string
 3522 shall be written.

3523 % Write a '%' character; no argument is converted.

3524 In no case does a nonexistent or insufficient *field width* cause truncation of a field; if the result of
 3525 a conversion is wider than the field width, the field is simply expanded to contain the conversion
 3526 result. The term *field width* should not be confused with the term *precision* used in the description
 3527 of %s.

3528 Examples

3529 To represent the output of a program that prints a date and time in the form Sunday, July 3,
 3530 10:02, where *weekday* and *month* are strings:

3531 "%s, Δ%s Δ%d, Δ%d: %.2d\n" <weekday>, <month>, <day>, <hour>, <min>

3532 To show 'π' written to 5 decimal places:

3533 "pi Δ= Δ%.5f\n", <value of π>

3534 To show an input file format consisting of five colon-separated fields:

3535 "%s: %s: %s: %s: %s\n", <arg1>, <arg2>, <arg3>, <arg4>, <arg5>

Character Set

3536

3537 6.1 Portable Character Set

3538 Conforming implementations shall support one or more coded character sets. Each supported
 3539 locale shall include the *portable character set*, which is the set of symbolic names for characters in
 3540 Table 6-1. This is used to describe characters within the text of IEEE Std 1003.1-200x. The first
 3541 eight entries in Table 6-1 are defined in the ISO/IEC 6429:1992 standard and the rest of the
 3542 characters are defined in the ISO/IEC 10646-1:2000 standard.

3543 **Table 6-1** Portable Character Set

3544

3545

3546

3547

3548

3549

3550

3551

3552

3553

3554

3555

3556

3557

3558

3559

3560

3561

3562

3563

3564

3565

3566

3567

3568

3569

3570

3571

3572

3573

3574

3575

3576

Symbolic Name	Glyph	UCS	Description
<NUL>		<U0000>	NULL (NUL)
<alert>		<U0007>	BELL (BEL)
<backspace>		<U0008>	BACKSPACE (BS)
<tab>		<U0009>	CHARACTER TABULATION (HT)
<carriage-return>		<U000D>	CARRIAGE RETURN (CR)
<newline>		<U000A>	LINE FEED (LF)
<vertical-tab>		<U000B>	LINE TABULATION (VT)
<form-feed>		<U000C>	FORM FEED (FF)
<space>		<U0020>	SPACE
<exclamation-mark>	!	<U0021>	EXCLAMATION MARK
<quotation-mark>	"	<U0022>	QUOTATION MARK
<number-sign>	#	<U0023>	NUMBER SIGN
<dollar-sign>	\$	<U0024>	DOLLAR SIGN
<percent-sign>	%	<U0025>	PERCENT SIGN
<ampersand>	&	<U0026>	AMPERSAND
<apostrophe>	'	<U0027>	APOSTROPHE
<left-parenthesis>	(<U0028>	LEFT PARENTHESIS
<right-parenthesis>)	<U0029>	RIGHT PARENTHESIS
<asterisk>	*	<U002A>	ASTERISK
<plus-sign>	+	<U002B>	PLUS SIGN
<comma>	,	<U002C>	COMMA
<hyphen-minus>	-	<U002D>	HYPHEN-MINUS
<hyphen>	-	<U002D>	HYPHEN-MINUS
<full-stop>	.	<U002E>	FULL STOP
<period>	.	<U002E>	FULL STOP
<slash>	/	<U002F>	SOLIDUS
<solidus>	/	<U002F>	SOLIDUS
<zero>	0	<U0030>	DIGIT ZERO
<one>	1	<U0031>	DIGIT ONE
<two>	2	<U0032>	DIGIT TWO
<three>	3	<U0033>	DIGIT THREE

3577
3578
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3620
3621
3622
3623
3624
3625

Symbolic Name	Glyph	UCS	Description
<four>	4	<U0034>	DIGIT FOUR
<five>	5	<U0035>	DIGIT FIVE
<six>	6	<U0036>	DIGIT SIX
<seven>	7	<U0037>	DIGIT SEVEN
<eight>	8	<U0038>	DIGIT EIGHT
<nine>	9	<U0039>	DIGIT NINE
<colon>	:	<U003A>	COLON
<semicolon>	;	<U003B>	SEMICOLON
<less-than-sign>	<	<U003C>	LESS-THAN SIGN
<equals-sign>	=	<U003D>	EQUALS SIGN
<greater-than-sign>	>	<U003E>	GREATER-THAN SIGN
<question-mark>	?	<U003F>	QUESTION MARK
<commercial-at>	@	<U0040>	<U0040>
<A>	A	<U0041>	LATIN CAPITAL LETTER A
	B	<U0042>	LATIN CAPITAL LETTER B
<C>	C	<U0043>	LATIN CAPITAL LETTER C
<D>	D	<U0044>	LATIN CAPITAL LETTER D
<E>	E	<U0045>	LATIN CAPITAL LETTER E
<F>	F	<U0046>	LATIN CAPITAL LETTER F
<G>	G	<U0047>	LATIN CAPITAL LETTER G
<H>	H	<U0048>	LATIN CAPITAL LETTER H
<I>	I	<U0049>	LATIN CAPITAL LETTER I
<J>	J	<U004A>	LATIN CAPITAL LETTER J
<K>	K	<U004B>	LATIN CAPITAL LETTER K
<L>	L	<U004C>	LATIN CAPITAL LETTER L
<M>	M	<U004D>	LATIN CAPITAL LETTER M
<N>	N	<U004E>	LATIN CAPITAL LETTER N
<O>	O	<U004F>	LATIN CAPITAL LETTER O
<P>	P	<U0050>	LATIN CAPITAL LETTER P
<Q>	Q	<U0051>	LATIN CAPITAL LETTER Q
<R>	R	<U0052>	LATIN CAPITAL LETTER R
<S>	S	<U0053>	LATIN CAPITAL LETTER S
<T>	T	<U0054>	LATIN CAPITAL LETTER T
<U>	U	<U0055>	LATIN CAPITAL LETTER U
<V>	V	<U0056>	LATIN CAPITAL LETTER V
<W>	W	<U0057>	LATIN CAPITAL LETTER W
<X>	X	<U0058>	LATIN CAPITAL LETTER X
<Y>	Y	<U0059>	LATIN CAPITAL LETTER Y
<Z>	Z	<U005A>	LATIN CAPITAL LETTER Z
<left-square-bracket>	[<U005B>	LEFT SQUARE BRACKET
<backslash>	\	<U005C>	REVERSE SOLIDUS
<reverse-solidus>	\	<U005C>	REVERSE SOLIDUS
<right-square-bracket>]	<U005D>	RIGHT SQUARE BRACKET
<circumflex-accent>	^	<U005E>	CIRCUMFLEX ACCENT
<circumflex>	^	<U005E>	CIRCUMFLEX ACCENT
<low-line>	_	<U005F>	LOW LINE
<underscore>	_	<U005F>	LOW LINE

3626

3627

3628

3629

3630

3631

3632

3633

3634

3635

3636

3637

3638

3639

3640

3641

3642

3643

3644

3645

3646

3647

3648

3649

3650

3651

3652

3653

3654

3655

3656

3657

3658

3659

3660

Symbolic Name	Glyph	UCS	Description
<grave-accent>	`	<U0060>	GRAVE ACCENT
<a>	a	<U0061>	LATIN SMALL LETTER A
	b	<U0062>	LATIN SMALL LETTER B
<c>	c	<U0063>	LATIN SMALL LETTER C
<d>	d	<U0064>	LATIN SMALL LETTER D
<e>	e	<U0065>	LATIN SMALL LETTER E
<f>	f	<U0066>	LATIN SMALL LETTER F
<g>	g	<U0067>	LATIN SMALL LETTER G
<h>	h	<U0068>	LATIN SMALL LETTER H
<i>	i	<U0069>	LATIN SMALL LETTER I
<j>	j	<U006A>	LATIN SMALL LETTER J
<k>	k	<U006B>	LATIN SMALL LETTER K
<l>	l	<U006C>	LATIN SMALL LETTER L
<m>	m	<U006D>	LATIN SMALL LETTER M
<n>	n	<U006E>	LATIN SMALL LETTER N
<o>	o	<U006F>	LATIN SMALL LETTER O
<p>	p	<U0070>	LATIN SMALL LETTER P
<q>	q	<U0071>	LATIN SMALL LETTER Q
<r>	r	<U0072>	LATIN SMALL LETTER R
<s>	s	<U0073>	LATIN SMALL LETTER S
<t>	t	<U0074>	LATIN SMALL LETTER T
<u>	u	<U0075>	LATIN SMALL LETTER U
<v>	v	<U0076>	LATIN SMALL LETTER V
<w>	w	<U0077>	LATIN SMALL LETTER W
<x>	x	<U0078>	LATIN SMALL LETTER X
<y>	y	<U0079>	LATIN SMALL LETTER Y
<z>	z	<U007A>	LATIN SMALL LETTER Z
<left-brace>	{	<U007B>	LEFT CURLY BRACKET
<left-curly-bracket>	{	<U007B>	LEFT CURLY BRACKET
<vertical-line>		<U007C>	VERTICAL LINE
<right-brace>	}	<U007D>	RIGHT CURLY BRACKET
<right-curly-bracket>	}	<U007D>	RIGHT CURLY BRACKET
<tilde>	~	<U007E>	TILDE

3661

3662

3663

IEEE Std 1003.1-200x uses character names other than the above, but only in an informative way; for example, in examples to illustrate the use of characters beyond the portable character set with the facilities of IEEE Std 1003.1-200x.

3664

3665

3666

3667

3668

Table 6-1 (on page 111) defines the characters in the portable character set and the corresponding symbolic character names used to identify each character in a character set description file. The table contains more than one symbolic character name for characters whose traditional name differs from the chosen name. Characters defined in Table 6-2 (on page 116) may also be used in character set description files.

3669

3670

IEEE Std 1003.1-200x places only the following requirements on the encoded values of the characters in the portable character set:

3671

3672

3673

3674

3675

- If the encoded values associated with each member of the portable character set are not invariant across all locales supported by the implementation, if an application accesses any pair of locales where the character encodings differ, or accesses data from an application running in a locale which has different encodings from the application's current locale, the results are unspecified.

- 3676 • The encoded values associated with the digits 0 to 9 shall be such that the value of each
3677 character after 0 shall be one greater than the value of the previous character.
- 3678 • A null character, NUL, which has all bits set to zero, shall be in the set of characters.
- 3679 • The encoded values associated with the members of the portable character set are each
3680 represented in a single byte. Moreover, if the value is stored in an object of C-language type
3681 **char**, it is guaranteed to be positive (except the NUL, which is always zero).
- 3682 Conforming implementations shall support certain character and character set attributes, as
3683 defined in Section 7.2 (on page 120).

3684 **6.2 Character Encoding**

3685 The POSIX locale contains the characters in Table 6-1 (on page 111), which have the properties
3686 listed in Section 7.3.1 (on page 122). In other locales, the presence, meaning, and representation
3687 of any additional characters is locale-specific.

3688 In locales other than the POSIX locale, a character may have a state-dependent encoding. There
3689 are two types of these encodings:

- 3690 • A single-shift encoding (where each character not in the initial shift state is preceded by a
3691 shift code) can be defined if each shift-code and character sequence is considered a multi-
3692 byte character. This is done using the concatenated-constant format in a character set
3693 description file, as described in Section 6.4 (on page 115). If the implementation supports a
3694 character encoding of this type, all of the standard utilities in the Shell and Utilities volume of |
3695 IEEE Std 1003.1-200x shall support it. Use of a single-shift encoding with any of the functions |
3696 in the System Interfaces volume of IEEE Std 1003.1-200x that do not specifically mention the |
3697 effects of state-dependent encoding is implementation-defined.
- 3698 • A locking-shift encoding (where the state of the character is determined by a shift code that
3699 may affect more than the single character following it) cannot be defined with the current
3700 character set description file format. Use of a locking-shift encoding with any of the standard
3701 utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x or with any of the functions
3702 in the System Interfaces volume of IEEE Std 1003.1-200x that do not specifically mention the
3703 effects of state-dependent encoding is implementation-defined.

3704 While in the initial shift state, all characters in the portable character set shall retain their usual |
3705 interpretation and shall not alter the shift state. The interpretation for subsequent bytes in the |
3706 sequence shall be a function of the current shift state. A byte with all bits zero shall be |
3707 interpreted as the null character independent of shift state. Thus a byte with all bits zero shall |
3708 never occur in the second or subsequent bytes of a character. |

3709 The maximum allowable number of bytes in a character in the current locale shall be indicated |
3710 by {MB_CUR_MAX}, defined in the `<stdlib.h>` header and by the `<mb_cur_max>` value in a |
3711 character set description file; see Section 6.4 (on page 115). The implementation's maximum |
3712 number of bytes in a character shall be defined by the C-language macro {MB_LEN_MAX}. |

3713 **6.3 C Language Wide-Character Codes**

3714 In the shell, the standard utilities are written so that the encodings of characters are described by
 3715 the locale's *LC_CTYPE* definition (see Section 7.3.1 (on page 122)) and there is no differentiation
 3716 between characters consisting of single octets (8-bit bytes) or multiple bytes. However, in the C
 3717 language, a differentiation is made. To ease the handling of variable length characters, the C
 3718 language has introduced the concept of wide-character codes.

3719 All wide-character codes in a given process consist of an equal number of bits. This is in contrast
 3720 to characters, which can consist of a variable number of bytes. The byte or byte sequence that
 3721 represents a character can also be represented as a wide-character code. Wide-character codes
 3722 thus provide a uniform size for manipulating text data. A wide-character code having all bits
 3723 zero is the null wide-character code (see Section 3.246 (on page 66)), and terminates wide-
 3724 character strings (see Section 3.432 (on page 92)). The wide-character value for each member of
 3725 the portable character set shall equal its value when used as the lone character in an integer
 3726 character constant. Wide-character codes for other characters are locale and implementation-
 3727 defined. State shift bytes shall not have a wide-character code representation.

3728 **6.4 Character Set Description File**

3729 Implementations shall provide a character set description file for at least one coded character set
 3730 supported by the implementation. These files are referred to elsewhere in IEEE Std 1003.1-200x
 3731 as *charmap* files. It is implementation-defined whether or not users or applications can provide
 3732 additional character set description files.

3733 IEEE Std 1003.1-200x does not require that multiple character sets or codesets be supported.
 3734 Although multiple charmap files are supported, it is the responsibility of the implementation to
 3735 provide the file or files; if only one is provided, only that one is accessible using the *localedef*
 3736 utility's *-f* option.

3737 Each character set description file, except those that use the ISO/IEC 10646-1:2000 standard
 3738 position values as the encoding values, shall define characteristics for the coded character set
 3739 and the encoding for the characters specified in Table 6-1 (on page 111), and may define
 3740 encoding for additional characters supported by the implementation. Other information about
 3741 the coded character set may also be in the file. Coded character set character values shall be
 3742 defined using symbolic character names followed by character encoding values.

3743 Each symbolic name specified in Table 6-1 (on page 111) shall be included in the file and shall be
 3744 mapped to a unique coding value, except as noted below. The glyphs '{', '}', '_-', '/',
 3745 '\', '.', and '^' have more than one symbolic name; all symbolic names for each such glyph
 3746 shall be included, each with identical encoding. If some or all of the control characters identified
 3747 in Table 6-2 (on page 116) are supported by the implementation, the symbolic names and their
 3748 corresponding encoding values shall be included in the file. Some of the encodings associated
 3749 with the symbolic names in Table 6-2 (on page 116) may be the same as characters found in Table
 3750 6-1 (on page 111); both names shall be provided for each encoding.

3751

Table 6-2 Control Character Set

3752	<ACK>	<DC2>	<ENQ>	<FS>	<IS4>	<SOH>
3753	<BEL>	<DC3>	<EOT>	<GS>	<LF>	<STX>
3754	<BS>	<DC4>	<ESC>	<HT>	<NAK>	<SUB>
3755	<CAN>		<ETB>	<IS1>	<RS>	<SYN>
3756	<CR>	<DLE>	<ETX>	<IS2>	<SI>	<US>
3757	<DC1>		<FF>	<IS3>	<SO>	<VT>

3758 The following declarations can precede the character definitions. Each shall consist of the
 3759 symbol shown in the following list, starting in column 1, including the surrounding brackets,
 3760 followed by one or more <blank>s, followed by the value to be assigned to the symbol.

3761 <code_set_name> The name of the coded character set for which the character set
 3762 description file is defined. The characters of the name shall be taken from
 3763 the set of characters with visible glyphs defined in Table 6-1 (on page
 3764 111).

3765 <mb_cur_max> The maximum number of bytes in a multi-byte character. This shall
 3766 default to 1.

3767 <mb_cur_min> An unsigned positive integer value that defines the minimum number of
 3768 XSI bytes in a character for the encoded character set. On XSI-conformant
 3769 systems, <mb_cur_min> shall always be 1.

3770 <escape_char> The character used to indicate that the characters following shall be
 3771 interpreted in a special way, as defined later in this section. This shall
 3772 default to backslash ('\''), which is the character used in all the following
 3773 text and examples, unless otherwise noted.

3774 <comment_char> The character that, when placed in column 1 of a charmap line, is used to
 3775 indicate that the line shall be ignored. The default character shall be the
 3776 number sign ('#').

3777 The character set mapping definitions shall be all the lines immediately following an identifier
 3778 line containing the string "CHARMAP" starting in column 1, and preceding a trailer line
 3779 containing the string "END CHARMAP" starting in column 1. Empty lines and lines containing a
 3780 <comment_char> in the first column shall be ignored. Each non-comment line of the character
 3781 set mapping definition (that is, between the "CHARMAP" and "END CHARMAP" lines of the file)
 3782 shall be in either of two forms:

3783 "%s %s %s\n", <symbolic-name>, <encoding>, <comments>

3784 or:

3785 "%s...%s %s %s\n", <symbolic-name>, <symbolic-name>,
 3786 <encoding>, <comments>

3787 In the first format, the line in the character set mapping definition shall define a single symbolic
 3788 name and a corresponding encoding. A symbolic name is one or more characters from the set
 3789 shown with visible glyphs in Table 6-1 (on page 111), enclosed between angle brackets. A
 3790 character following an escape character is interpreted as itself; for example, the sequence
 3791 "<\\>" represents the symbolic name ">" enclosed between angle brackets.

3792 In the second format, the line in the character set mapping definition shall define a range of one
 3793 or more symbolic names. In this form, the symbolic names shall consist of zero or more non-
 3794 numeric characters from the set shown with visible glyphs in Table 6-1 (on page 111), followed
 3795 by an integer formed by one or more decimal digits. Both integers shall contain the same number
 3796 of digits. The characters preceding the integer shall be identical in the two symbolic names, and

3797 the integer formed by the digits in the second symbolic name shall be equal to or greater than the
 3798 integer formed by the digits in the first name. This shall be interpreted as a series of symbolic
 3799 names formed from the common part and each of the integers between the first and the second
 3800 integer, inclusive. As an example, <j0101>...<j0104> is interpreted as the symbolic names
 3801 <j0101>, <j0102>, <j0103>, and <j0104>, in that order.

3802 A character set mapping definition line shall exist for all symbolic names specified in Table 6-1
 3803 (on page 111), and shall define the coded character value that corresponds to the character
 3804 indicated in the table, or the coded character value that corresponds to the control character
 3805 symbolic name. If the control characters commonly associated with the symbolic names in Table
 3806 6-2 (on page 116) are supported by the implementation, the symbolic name and the
 3807 corresponding encoding value shall be included in the file. Additional unique symbolic names
 3808 may be included. A coded character value can be represented by more than one symbolic name.

3809 The encoding part is expressed as one (for single-byte character values) or more concatenated
 3810 decimal, octal, or hexadecimal constants in the following formats:

```
3811     "%cd%u", <escape_char>, <decimal byte value>
3812     "%cx%x", <escape_char>, <hexadecimal byte value>
3813     "%co", <escape_char>, <octal byte value>
```

3814 Decimal constants shall be represented by two or three decimal digits, preceded by the escape
 3815 character and the lowercase letter 'd'; for example, "\d05", "\d97", or "\d143".
 3816 Hexadecimal constants shall be represented by two hexadecimal digits, preceded by the escape
 3817 character and the lowercase letter 'x'; for example, "\x05", "\x61", or "\x8f". Octal
 3818 constants shall be represented by two or three octal digits, preceded by the escape character; for
 3819 example, "\05", "\141", or "\217". In a portable charmap file, each constant represents an 8-
 3820 bit byte. When constants are concatenated for multi-byte character values, they shall be of the
 3821 same type, and interpreted in byte order from first to last with the least significant byte of the
 3822 multi-byte character specified by the last constant. The manner in which these constants are
 3823 represented in the character stored in the system is implementation-defined. (This notation was
 3824 chosen for reasons of portability. There is no requirement that the internal representation in the
 3825 computer memory be in this same order.) Omitting bytes from a multi-byte character definition
 3826 produces undefined results.

3827 In lines defining ranges of symbolic names, the encoded value shall be the value for the first
 3828 symbolic name in the range (the symbolic name preceding the ellipsis). Subsequent symbolic
 3829 names defined by the range shall have encoding values in increasing order. Bytes shall be treated
 3830 as unsigned octets, and carry shall be propagated between the bytes as necessary to represent
 3831 the range. For example, the line:

```
3832     <j0101>...<j0104>  \d129\d254
```

3833 is interpreted as:

```
3834     <j0101>             \d129\d254
3835     <j0102>             \d129\d255
3836     <j0103>             \d130\d0
3837     <j0104>             \d130\d1
```

3838 Note that this line is interpreted as the example even on systems with bytes larger than 8 bits.

3839 The comment is optional.

3840 The following declarations can follow the character set mapping definitions (after the "END
 3841 CHARMAP" statement). Each shall consist of the keyword shown in the following list, starting in
 3842 column 1, followed by the value(s) to be associated to the keyword, as defined below.

3843 **WIDTH** An unsigned positive integer value defining the column width (see Section 3.103
 3844 (on page 47)) for the printable characters in the coded character set specified in
 3845 Table 6-1 (on page 111) and Table 6-2 (on page 116). Coded character set character
 3846 values shall be defined using symbolic character names followed by column width
 3847 values. Defining a character with more than one **WIDTH** produces undefined
 3848 results. The **END WIDTH** keyword shall be used to terminate the **WIDTH**
 3849 definitions. Specifying the width of a non-printable character in a **WIDTH**
 3850 declaration produces undefined results.

3851 **WIDTH_DEFAULT**
 3852 An unsigned positive integer value defining the default column width for any
 3853 printable character not listed by one of the **WIDTH** keywords. If no
 3854 **WIDTH_DEFAULT** keyword is included in the charmap, the default character
 3855 width shall be 1.

3856 **Example**

3857 After the "END CHARMAP" statement, a syntax for a width definition would be:

```
3858            WIDTH
3859            <A> 1
3860            <B> 1
3861            <C>...<Z> 1
3862            <fool>...<foon> 2
3863            END WIDTH
```

3864 In this example, the numerical code point values represented by the symbols <A> and are
 3865 assigned a width of 1. The code point values <C> to <Z> inclusive (<C>, <D>, <E>, and so on)
 3866 are also assigned a width of 1. Using <A>...<Z> would have required fewer lines, but the
 3867 alternative was shown to demonstrate flexibility. The keyword **WIDTH_DEFAULT** could have
 3868 been added as appropriate.

3869 **6.4.1 State-Dependent Character Encodings**

3870 This section addresses the use of state-dependent character encodings (that is, those in which the
 3871 encoding of a character is dependent on one or more shift codes that may precede it).

3872 A single-shift encoding (where each character not in the initial shift state is preceded by a shift
 3873 code) can be defined in the charmap format if each shift-code/character sequence is considered a
 3874 multi-byte character, defined using the concatenated-constant format described in Section 6.4
 3875 (on page 115). If the implementation supports a character encoding of this type, all of the
 3876 standard utilities shall support it. A locking-shift encoding (where the state of the character is
 3877 determined by a shift code that may affect more than the single character following it) could be
 3878 defined with an extension to the charmap format described in Section 6.4 (on page 115). If the
 3879 implementation supports a character encoding of this type, any of the standard utilities that
 3880 describe character (*versus* byte) or text-file manipulation shall have the following characteristics:

- 3881 1. The utility shall process the statefully encoded data as a concatenation of state-
 3882 independent characters. The presence of redundant locking shifts shall not affect the
 3883 comparison of two statefully encoded strings.
- 3884 2. A utility that divides, truncates, or extracts substrings from statefully encoded data shall
 3885 produce output that contains locking shifts at the beginning or end of the resulting data, if
 3886 appropriate, to retain correct state information.

3888 **7.1 General**

3889 A *locale* is the definition of the subset of a user's environment that depends on language and
 3890 cultural conventions. It is made up from one or more categories. Each category is identified by
 3891 its name and controls specific aspects of the behavior of components of the system. Category
 3892 names correspond to the following environment variable names:

3893 *LC_CTYPE* Character classification and case conversion.

3894 *LC_COLLATE* Collation order.

3895 *LC_MONETARY* Monetary formatting.

3896 *LC_NUMERIC* Numeric, non-monetary formatting.

3897 *LC_TIME* Date and time formats.

3898 *LC_MESSAGES* Formats of informative and diagnostic messages and interactive responses.

3899 The standard utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x shall base their
 3900 behavior on the current locale, as defined in the ENVIRONMENT VARIABLES section for each
 3901 utility. The behavior of some of the C-language functions defined in the System Interfaces
 3902 volume of IEEE Std 1003.1-200x shall also be modified based on the current locale, as defined by
 3903 the last call to *setlocale()*.

3904 Locales other than those supplied by the implementation can be created via the *localedef* utility,
 3905 provided that the *_POSIX2_LOCALEDEF* symbol is defined on the system. Even if *localedef* is not
 3906 provided, all implementations conforming to the System Interfaces volume of
 3907 IEEE Std 1003.1-200x shall provide one or more locales that behave as described in this chapter.
 3908 The input to the utility is described in Section 7.3 (on page 120). The value that is used to specify
 3909 a locale when using environment variables shall be the string specified as the *name* operand to
 3910 the *localedef* utility when the locale was created. The strings "C" and "POSIX" are reserved as
 3911 identifiers for the POSIX locale (see Section 7.2 (on page 120)). When the value of a locale
 3912 environment variable begins with a slash ('/'), it shall be interpreted as the pathname of the
 3913 locale definition; the type of file (regular, directory, and so on) used to store the locale definition
 3914 is implementation-defined. If the value does not begin with a slash, the mechanism used to
 3915 locate the locale is implementation-defined.

3916 If different character sets are used by the locale categories, the results achieved by an application
 3917 utilizing these categories are undefined. Likewise, if different codesets are used for the data
 3918 being processed by interfaces whose behavior is dependent on the current locale, or the codeset
 3919 is different from the codeset assumed when the locale was created, the result is also undefined.

3920 Applications can select the desired locale by invoking the *setlocale()* function (or equivalent)
 3921 with the appropriate value. If the function is invoked with an empty string, such as:

```
3922     setlocale(LC_ALL, "");
```

3923 the value of the corresponding environment variable is used. If the environment variable is
 3924 unset or is set to the empty string, the implementation shall set the appropriate environment as
 3925 defined in Chapter 8 (on page 157).

3926 7.2 POSIX Locale

3927 Conforming systems shall provide a *POSIX locale*, also known as the C locale. The behavior of
3928 standard utilities and functions in the POSIX locale shall be as if the locale was defined via the
3929 *localedef* utility with input data from the POSIX locale tables in Section 7.3.

3930 The tables in Section 7.3 describe the characteristics and behavior of the POSIX locale for data
3931 consisting entirely of characters from the portable character set and the control character set. For
3932 other characters, the behavior is unspecified. For C-language programs, the POSIX locale shall be
3933 the default locale when the *setlocale()* function is not called.

3934 The POSIX locale can be specified by assigning to the appropriate environment variables the
3935 values "C" or "POSIX".

3936 All implementations shall define a locale as the default locale, to be invoked when no
3937 environment variables are set, or set to the empty string. This default locale can be the POSIX
3938 locale or any other implementation-defined locale. Some implementations may provide facilities
3939 for local installation administrators to set the default locale, customizing it for each location.
3940 IEEE Std 1003.1-200x does not require such a facility.

3941 7.3 Locale Definition

3942 The capability to specify additional locales to those provided by an implementation is optional,
3943 denoted by the `_POSIX2_LOCALEDEF` symbol. If the option is not supported, only
3944 implementation-supplied locales are available. Such locales shall be documented using the
3945 format specified in this section.

3946 Locales can be described with the file format presented in this section. The file format is that
3947 accepted by the *localedef* utility. For the purposes of this section, the file is referred to as the *locale
3948 definition file*, but no locales shall be affected by this file unless it is processed by *localedef* or some
3949 similar mechanism. Any requirements in this section imposed upon the utility shall apply to
3950 *localedef* or to any other similar utility used to install locale information using the locale
3951 definition file format described here.

3952 The locale definition file shall contain one or more locale category source definitions, and shall
3953 not contain more than one definition for the same locale category. If the file contains source
3954 definitions for more than one category, implementation-defined categories, if present, shall
3955 appear after the categories defined by Section 7.1 (on page 119). A category source definition
3956 contains either the definition of a category or a **copy** directive. For a description of the **copy**
3957 directive, see *localedef*. In the event that some of the information for a locale category, as
3958 specified in this volume of IEEE Std 1003.1-200x, is missing from the locale source definition, the
3959 behavior of that category, if it is referenced, is unspecified.

3960 A category source definition shall consist of a category header, a category body, and a category
3961 trailer. A category header shall consist of the character string naming of the category, beginning
3962 with the characters `LC_`. The category trailer shall consist of the string "END", followed by one
3963 or more `<blank>`s and the string used in the corresponding category header.

3964 The category body shall consist of one or more lines of text. Each line shall contain an identifier,
3965 optionally followed by one or more operands. Identifiers shall be either keywords, identifying a
3966 particular locale element, or collating elements. In addition to the keywords defined in this
3967 volume of IEEE Std 1003.1-200x, the source can contain implementation-defined keywords. Each
3968 keyword within a locale shall have a unique name (that is, two categories cannot have a
3969 commonly-named keyword); no keyword shall start with the characters `LC_`. Identifiers shall be
3970 separated from the operands by one or more `<blank>`s.

3971 Operands shall be characters, collating elements, or strings of characters. Strings shall be
 3972 enclosed in double-quotes. Literal double-quotes within strings shall be preceded by the *<escape*
 3973 *character>*, described below. When a keyword is followed by more than one operand, the
 3974 operands shall be separated by semicolons; *<blank>*s shall be allowed both before and after a
 3975 semicolon.

3976 The first category header in the file can be preceded by a line modifying the comment character.
 3977 It shall have the following format, starting in column 1:

```
3978     "comment_char %c\n", <comment character>
```

3979 The comment character shall default to the number sign ('#'). Blank lines and lines containing
 3980 the *<comment character>* in the first position shall be ignored.

3981 The first category header in the file can be preceded by a line modifying the escape character to
 3982 be used in the file. It shall have the following format, starting in column 1:

```
3983     "escape_char %c\n", <escape character>
```

3984 The escape character shall default to backslash, which is the character used in all examples
 3985 shown in this volume of IEEE Std 1003.1-200x.

3986 A line can be continued by placing an escape character as the last character on the line; this
 3987 continuation character shall be discarded from the input. Although the implementation need not
 3988 accept any one portion of a continued line with a length exceeding {LINE_MAX} bytes, it shall
 3989 place no limits on the accumulated length of the continued line. Comment lines shall not be
 3990 continued on a subsequent line using an escaped newline character.

3991 Individual characters, characters in strings, and collating elements shall be represented using
 3992 symbolic names, as defined below. In addition, characters can be represented using the
 3993 characters themselves or as octal, hexadecimal, or decimal constants. When non-symbolic
 3994 notation is used, the resultant locale definitions are in many cases not portable between systems.
 3995 The left angle bracket ('<') is a reserved symbol, denoting the start of a symbolic name; when
 3996 used to represent itself it shall be preceded by the escape character. The following rules apply to
 3997 character representation:

- 3998 1. A character can be represented via a symbolic name, enclosed within angle brackets '<'
 3999 and '>'. The symbolic name, including the angle brackets, shall exactly match a symbolic
 4000 name defined in the charmap file specified via the *localedef -f* option, and it shall be
 4001 replaced by a character value determined from the value associated with the symbolic
 4002 name in the charmap file. The use of a symbolic name not found in the charmap file shall
 4003 constitute an error, unless the category is *LC_CTYPE* or *LC_COLLATE*, in which case it
 4004 shall constitute a warning condition (see *localedef* for a description of actions resulting from
 4005 errors and warnings). The specification of a symbolic name in a **collating-element** or
 4006 **collating-symbol** section that duplicates a symbolic name in the charmap file (if present)
 4007 shall be an error. Use of the escape character or a right angle bracket within a symbolic
 4008 name is invalid unless the character is preceded by the escape character.

4009 For example:

```
4010     <c>;<c-cedilla>    "<M><a><y>"
```

- 4011 2. A character in the portable character set can be represented by the character itself, in which
 4012 case the value of the character is implementation-defined. (Implementations may allow
 4013 other characters to be represented as themselves, but such locale definitions are not
 4014 portable.) Within a string, the double-quote character, the escape character, and the right
 4015 angle bracket character shall be escaped (preceded by the escape character) to be
 4016 interpreted as the character itself. Outside strings, the characters:

4017 , ; < > *escape_char*

4018 shall be escaped to be interpreted as the character itself.

4019 For example:

4020 c "May"

4021 3. A character can be represented as an octal constant. An octal constant shall be specified as
4022 the escape character followed by two or three octal digits. Each constant shall represent a
4023 byte value. Multi-byte values can be represented by concatenated constants specified in
4024 byte order with the last constant specifying the least significant byte of the character.

4025 For example:

4026 \143;\347;\143\150 "\115\141\171"

4027 4. A character can be represented as a hexadecimal constant. A hexadecimal constant shall be
4028 specified as the escape character followed by an 'x' followed by two hexadecimal digits.
4029 Each constant shall represent a byte value. Multi-byte values can be represented by
4030 concatenated constants specified in byte order with the last constant specifying the least
4031 significant byte of the character.

4032 For example:

4033 \x63;\xe7;\x63\x68 "\x4d\x61\x79"

4034 5. A character can be represented as a decimal constant. A decimal constant shall be specified
4035 as the escape character followed by a 'd' followed by two or three decimal digits. Each
4036 constant represents a byte value. Multi-byte values can be represented by concatenated
4037 constants specified in byte order with the last constant specifying the least significant byte
4038 of the character.

4039 For example:

4040 \d99;\d231;\d99\d104 "\d77\d97\d121"

4041 Implementations may accept single-digit octal, decimal, or hexadecimal constants following the
4042 escape character. Only characters existing in the character set for which the locale definition is
4043 created shall be specified, whether using symbolic names, the characters themselves, or octal,
4044 decimal, or hexadecimal constants. If a charmap file is present, only characters defined in the
4045 charmap can be specified using octal, decimal, or hexadecimal constants. Symbolic names not
4046 present in the charmap file can be specified and shall be ignored, as specified under item 1
4047 above.

4048 7.3.1 LC_CTYPE

4049 The *LC_CTYPE* category shall define character classification, case conversion, and other
4050 character attributes. In addition, a series of characters can be represented by three adjacent
4051 periods representing an ellipsis symbol ("..."). The ellipsis specification shall be interpreted as
4052 meaning that all values between the values preceding and following it represent valid
4053 characters. The ellipsis specification shall be valid only within a single encoded character set;
4054 that is, within a group of characters of the same size. An ellipsis shall be interpreted as including
4055 in the list all characters with an encoded value higher than the encoded value of the character
4056 preceding the ellipsis and lower than the encoded value of the character following the ellipsis.

4057 For example:

4058 \x30;...;\x39;

4059 includes in the character class all characters with encoded values between the endpoints.

4060 The following keywords shall be recognized. In the descriptions, the term “automatically
4061 included” means that it shall not be an error either to include or omit any of the referenced
4062 characters; the implementation provides them if missing (even if the entire keyword is missing)
4063 and accepts them silently if present. When the implementation automatically includes a missing
4064 character, it shall have an encoded value dependent on the charmap file in effect (see the
4065 description of the *localedef* **-f** option); otherwise, it shall have a value derived from an
4066 implementation-defined character mapping.

4067 The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included
4068 characters. These only need to be specified if the character values (that is, encoding) differ from
4069 the implementation default values. It is not possible to define a locale without these
4070 automatically included characters unless some implementation extension is used to prevent
4071 their inclusion. Such a definition would not be a proper superset of the C or POSIX locale and
4072 thus, it might not be possible for conforming applications to work properly.

4073 **copy** Specify the name of an existing locale which shall be used as the definition of |
4074 this category. If this keyword is specified, no other keyword shall be specified. |

4075 **upper** Define characters to be classified as uppercase letters.
4076 In the POSIX locale, the 26 uppercase letters shall be included: |
4077 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

4078 In a locale definition file, no character specified for the keywords **cntrl**, **digit**,
4079 **punct**, or **space** shall be specified. The uppercase letters <A> to <Z>, as
4080 defined in Section 6.4 (on page 115) (the portable character set), are
4081 automatically included in this class.

4082 **lower** Define characters to be classified as lowercase letters.
4083 In the POSIX locale, the 26 lowercase letters shall be included: |
4084 a b c d e f g h i j k l m n o p q r s t u v w x y z

4085 In a locale definition file, no character specified for the keywords **cntrl**, **digit**,
4086 **punct**, or **space** shall be specified. The lowercase letters <a> to <z> of the
4087 portable character set are automatically included in this class.

4088 **alpha** Define characters to be classified as letters.
4089 In the POSIX locale, all characters in the classes **upper** and **lower** shall be |
4090 included. |

4091 In a locale definition file, no character specified for the keywords **cntrl**, **digit**,
4092 **punct**, or **space** shall be specified. Characters classified as either **upper** or
4093 **lower** are automatically included in this class.

4094 **digit** Define the characters to be classified as numeric digits.
4095 In the POSIX locale, only:
4096 0 1 2 3 4 5 6 7 8 9
4097 shall be included. |

4098 In a locale definition file, only the digits <zero>, <one>, <two>, <three>,
4099 <four>, <five>, <six>, <seven>, <eight>, and <nine> shall be specified, and in
4100 contiguous ascending sequence by numerical value. The digits <zero> to
4101 <nine> of the portable character set are automatically included in this class.

4102 4103 4104 4105	alnum	Define characters to be classified as letters and numeric digits. Only the characters specified for the alpha and digit keywords shall be specified. Characters specified for the keywords alpha and digit are automatically included in this class.
4106 4107 4108	space	Define characters to be classified as white-space characters. In the POSIX locale, at a minimum, the <space>, <form-feed>, <newline>, <carriage-return>, <tab>, and <vertical-tab> shall be included.
4109 4110 4111 4112 4113		In a locale definition file, no character specified for the keywords upper , lower , alpha , digit , graph , or xdigit shall be specified. The <space>, <form-feed>, <newline>, <carriage-return>, <tab>, and <vertical-tab> of the portable character set, and any characters included in the class blank are automatically included in this class.
4114 4115	cntrl	Define characters to be classified as control characters. In the POSIX locale, no characters in classes alpha or print shall be included.
4116 4117		In a locale definition file, no character specified for the keywords upper , lower , alpha , digit , punct , graph , print , or xdigit shall be specified.
4118 4119 4120	punct	Define characters to be classified as punctuation characters. In the POSIX locale, neither the <space> nor any characters in classes alpha , digit , or cntrl shall be included.
4121 4122		In a locale definition file, no character specified for the keywords upper , lower , alpha , digit , cntrl , xdigit , or as the <space> shall be specified.
4123 4124	graph	Define characters to be classified as printable characters, not including the <space>.
4125 4126		In the POSIX locale, all characters in classes alpha , digit , and punct shall be included; no characters in class cntrl shall be included.
4127 4128 4129		In a locale definition file, characters specified for the keywords upper , lower , alpha , digit , xdigit , and punct are automatically included in this class. No character specified for the keyword cntrl shall be specified.
4130 4131	print	Define characters to be classified as printable characters, including the <space>.
4132 4133		In the POSIX locale, all characters in class graph shall be included; no characters in class cntrl shall be included.
4134 4135 4136		In a locale definition file, characters specified for the keywords upper , lower , alpha , digit , xdigit , punct , graph , and the <space> are automatically included in this class. No character specified for the keyword cntrl shall be specified.
4137 4138	xdigit	Define the characters to be classified as hexadecimal digits. In the POSIX locale, only:
4139		0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f
4140		shall be included.
4141 4142 4143		In a locale definition file, only the characters defined for the class digit shall be specified, in contiguous ascending sequence by numerical value, followed by one or more sets of six characters representing the hexadecimal digits 10 to 15

4144		inclusive, with each set in ascending order (for example, <A>, , <C>, <D>, <E>, <F>, <a>, , <c>, <d>, <e>, <f>). The digits <zero> to <nine>, the uppercase letters <A> to <F>, and the lowercase letters <a> to <f> of the portable character set are automatically included in this class.
4145		
4146		
4147		
4148	blank	Define characters to be classified as <blank>s.
4149		In the POSIX locale, only the <space> and <tab> shall be included.
4150		In a locale definition file, the <space> and <tab> are automatically included in
4151		this class. LI charclass Define one or more locale-specific character class
4152		names as strings separated by semicolons. Each named character class can
4153		then be defined subsequently in the <i>LC_CTYPE</i> definition. A character class
4154		name shall consist of at least one and at most {CHARCLASS_NAME_MAX}
4155		bytes of alphanumeric characters from the portable filename character set. The
4156		first character of a character class name shall not be a digit. The name shall not
4157		match any of the <i>LC_CTYPE</i> keywords defined in this volume of
4158		IEEE Std 1003.1-200x. Future revisions of IEEE Std 1003.1-200x will not specify
4159		any <i>LC_CTYPE</i> keywords containing uppercase letters.
4160	<i>charclass-name</i>	Define characters to be classified as belonging to the named locale-specific
4161		character class. In the POSIX locale, locale-specific named character classes
4162		need not exist.
4163		If a class name is defined by a charclass keyword, but no characters are
4164		subsequently assigned to it, this is not an error; it represents a class without
4165		any characters belonging to it.
4166		The <i>charclass-name</i> can be used as the <i>property</i> argument to the <i>wctype()</i>
4167		function, in regular expression and shell pattern-matching bracket
4168		expressions, and by the <i>tr</i> command.
4169	toupper	Define the mapping of lowercase letters to uppercase letters.
4170		In the POSIX locale, at a minimum, the 26 lowercase characters:
4171		a b c d e f g h i j k l m n o p q r s t u v w x y z
4172		shall be mapped to the corresponding 26 uppercase characters:
4173		A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
4174		In a locale definition file, the operand shall consist of character pairs,
4175		separated by semicolons. The characters in each character pair shall be
4176		separated by a comma and the pair enclosed by parentheses. The first
4177		character in each pair is the lowercase letter, the second the corresponding
4178		uppercase letter. Only characters specified for the keywords lower and upper
4179		shall be specified. The lowercase letters <a> to <z>, and their corresponding
4180		uppercase letters <A> to <Z>, of the portable character set are automatically
4181		included in this mapping, but only when the toupper keyword is omitted
4182		from the locale definition.
4183	tolower	Define the mapping of uppercase letters to lowercase letters.
4184		In the POSIX locale, at a minimum, the 26 uppercase characters:
4185		A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
4186		shall be mapped to the corresponding 26 lowercase characters:

4187 a b c d e f g h i j k l m n o p q r s t u v w x y z

4188 In a locale definition file, the operand shall consist of character pairs, |
 4189 separated by semicolons. The characters in each character pair shall be |
 4190 separated by a comma and the pair enclosed by parentheses. The first |
 4191 character in each pair is the uppercase letter, the second the corresponding |
 4192 lowercase letter. Only characters specified for the keywords **lower** and **upper** |
 4193 shall be specified. If the **tolower** keyword is omitted from the locale definition,
 4194 the mapping is the reverse mapping of the one specified for **toupper**.

4195 The following table shows the character class combinations allowed:

4196 **Table 7-1** Valid Character Class Combinations

In Class	Can Also Belong To										
	upper	lower	alpha	digit	space	cntrl	punct	graph	print	xdigit	blank
4199 upper	—	—	A	x	x	x	x	A	A	—	x
4200 lower	—	—	A	x	x	x	x	A	A	—	x
4201 alpha	—	—	—	x	x	x	x	A	A	—	x
4202 digit	x	x	x	—	x	x	x	A	A	A	x
4203 space	x	x	x	x	—	—	*	*	*	x	—
4204 cntrl	x	x	x	x	—	—	x	x	x	x	—
4205 punct	x	x	x	x	—	x	—	A	A	x	—
4206 graph	—	—	—	—	—	x	—	—	A	—	—
4207 print	—	—	—	—	—	x	—	—	—	—	—
4208 xdigit	—	—	—	—	x	x	x	A	A	—	x
4209 blank	x	x	x	x	A	—	*	*	*	x	—

4210 **Notes:**

- 4211 1. Explanation of codes:
 4212 A Automatically included; see text.
 4213 — Permitted.
 4214 x Mutually-exclusive.
 4215 * See note 2.
- 4216 2. The <space>, which is part of the **space** and **blank** classes, cannot belong to **punct** or
 4217 **graph**, but shall automatically belong to the **print** class. Other **space** or **blank** characters
 4218 can be classified as any of **punct**, **graph**, or **print**.

4219 **7.3.1.1 LC_CTYPE Category in the POSIX Locale**

4220 The character classifications for the POSIX locale follow; the code listing depicts the *localedef*
 4221 input, the table represents the same information, sorted by character.

```

4222 LC_CTYPE
4223 # The following is the POSIX locale LC_CTYPE.
4224 # "alpha" is by default "upper" and "lower"
4225 # "alnum" is by definition "alpha" and "digit"
4226 # "print" is by default "alnum", "punct" and the <space>
4227 # "graph" is by default "alnum" and "punct"
4228 #
4229 upper <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;<M>;\
4230 <N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;<Z>
4231 #
    
```

```

4232 lower <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
4233 <n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>
4234 #
4235 digit <zero>;<one>;<two>;<three>;<four>;<five>;<six>;\
4236 <seven>;<eight>;<nine>
4237 #
4238 space <tab>;<newline>;<vertical-tab>;<form-feed>;\
4239 <carriage-return>;<space>
4240 #
4241 cntrl <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;\
4242 <form-feed>;<carriage-return>;\
4243 <NUL>;<SOH>;<STX>;<ETX>;<EOT>;<ENQ>;<ACK>;<SO>;\
4244 <SI>;<DLE>;<DC1>;<DC2>;<DC3>;<DC4>;<NAK>;<SYN>;\
4245 <ETB>;<CAN>;<EM>;<SUB>;<ESC>;<IS4>;<IS3>;<IS2>;\
4246 <IS1>;<DEL>
4247 #
4248 punct <exclamation-mark>;<quotation-mark>;<number-sign>;\
4249 <dollar-sign>;<percent-sign>;<ampersand>;<apostrophe>;\
4250 <left-parenthesis>;<right-parenthesis>;<asterisk>;\
4251 <plus-sign>;<comma>;<hyphen>;<period>;<slash>;\
4252 <colon>;<semicolon>;<less-than-sign>;<equals-sign>;\
4253 <greater-than-sign>;<question-mark>;<commercial-at>;\
4254 <left-square-bracket>;<backslash>;<right-square-bracket>;\
4255 <circumflex>;<underscore>;<grave-accent>;<left-curly-bracket>;\
4256 <vertical-line>;<right-curly-bracket>;<tilde>
4257 #
4258 xdigit <zero>;<one>;<two>;<three>;<four>;<five>;<six>;<seven>;\
4259 <eight>;<nine>;<A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
4260 #
4261 blank <space>;<tab>
4262 #
4263 toupper (<a>,<A>);(<b>,<B>);(<c>,<C>);(<d>,<D>);(<e>,<E>);\
4264 (<f>,<F>);(<g>,<G>);(<h>,<H>);(<i>,<I>);(<j>,<J>);\
4265 (<k>,<K>);(<l>,<L>);(<m>,<M>);(<n>,<N>);(<o>,<O>);\
4266 (<p>,<P>);(<q>,<Q>);(<r>,<R>);(<s>,<S>);(<t>,<T>);\
4267 (<u>,<U>);(<v>,<V>);(<w>,<W>);(<x>,<X>);(<y>,<Y>);(<z>,<Z>)
4268 #
4269 tolower (<A>,<a>);(<B>,<b>);(<C>,<c>);(<D>,<d>);(<E>,<e>);\
4270 (<F>,<f>);(<G>,<g>);(<H>,<h>);(<I>,<i>);(<J>,<j>);\
4271 (<K>,<k>);(<L>,<l>);(<M>,<m>);(<N>,<n>);(<O>,<o>);\
4272 (<P>,<p>);(<Q>,<q>);(<R>,<r>);(<S>,<s>);(<T>,<t>);\
4273 (<U>,<u>);(<V>,<v>);(<W>,<w>);(<X>,<x>);(<Y>,<y>);(<Z>,<z>)
4274 END LC_CTYPE

```

4275
4276
4277
4278
4279
4280
4281
4282
4283
4284
4285
4286
4287
4288
4289
4290
4291
4292
4293
4294
4295
4296
4297
4298
4299
4300
4301
4302
4303
4304
4305
4306
4307
4308
4309
4310
4311
4312
4313
4314
4315
4316
4317
4318
4319
4320
4321
4322
4323

Symbolic Name	Other Case	Character Classes
<NUL>		cntrl
<SOH>		cntrl
<STX>		cntrl
<ETX>		cntrl
<EOT>		cntrl
<ENQ>		cntrl
<ACK>		cntrl
<alert>		cntrl
<backspace>		cntrl
<tab>		cntrl, space, blank
<newline>		cntrl, space
<vertical-tab>		cntrl, space
<form-feed>		cntrl, space
<carriage-return>		cntrl, space
<SO>		cntrl
<SI>		cntrl
<DLE>		cntrl
<DC1>		cntrl
<DC2>		cntrl
<DC3>		cntrl
<DC4>		cntrl
<NAK>		cntrl
<SYN>		cntrl
<ETB>		cntrl
<CAN>		cntrl
		cntrl
<SUB>		cntrl
<ESC>		cntrl
<IS4>		cntrl
<IS3>		cntrl
<IS2>		cntrl
<IS1>		cntrl
<space>		space, print, blank
<exclamation-mark>		punct, print, graph
<quotation-mark>		punct, print, graph
<number-sign>		punct, print, graph
<dollar-sign>		punct, print, graph
<percent-sign>		punct, print, graph
<ampersand>		punct, print, graph
<apostrophe>		punct, print, graph
<left-parenthesis>		punct, print, graph
<right-parenthesis>		punct, print, graph
<asterisk>		punct, print, graph
<plus-sign>		punct, print, graph
<comma>		punct, print, graph
<hyphen>		punct, print, graph
<period>		punct, print, graph

4324
4325
4326
4327
4328
4329
4330
4331
4332
4333
4334
4335
4336
4337
4338
4339
4340
4341
4342
4343
4344
4345
4346
4347
4348
4349
4350
4351
4352
4353
4354
4355
4356
4357
4358
4359
4360
4361
4362
4363
4364
4365
4366
4367
4368
4369
4370
4371
4372

Symbolic Name	Other Case	Character Classes
<slash>		punct, print, graph
<zero>		digit, xdigit, print, graph
<one>		digit, xdigit, print, graph
<two>		digit, xdigit, print, graph
<three>		digit, xdigit, print, graph
<four>		digit, xdigit, print, graph
<five>		digit, xdigit, print, graph
<six>		digit, xdigit, print, graph
<seven>		digit, xdigit, print, graph
<eight>		digit, xdigit, print, graph
<nine>		digit, xdigit, print, graph
<colon>		punct, print, graph
<semicolon>		punct, print, graph
<less-than-sign>		punct, print, graph
<equals-sign>		punct, print, graph
<greater-than-sign>		punct, print, graph
<question-mark>		punct, print, graph
<commercial-at>		punct, print, graph
<A>	<a>	upper, xdigit, alpha, print, graph
		upper, xdigit, alpha, print, graph
<C>	<c>	upper, xdigit, alpha, print, graph
<D>	<d>	upper, xdigit, alpha, print, graph
<E>	<e>	upper, xdigit, alpha, print, graph
<F>	<f>	upper, xdigit, alpha, print, graph
<G>	<g>	upper, alpha, print, graph
<H>	<h>	upper, alpha, print, graph
<I>	<i>	upper, alpha, print, graph
<J>	<j>	upper, alpha, print, graph
<K>	<k>	upper, alpha, print, graph
<L>	<l>	upper, alpha, print, graph
<M>	<m>	upper, alpha, print, graph
<N>	<n>	upper, alpha, print, graph
<O>	<o>	upper, alpha, print, graph
<P>	<p>	upper, alpha, print, graph
<Q>	<q>	upper, alpha, print, graph
<R>	<r>	upper, alpha, print, graph
<S>	<s>	upper, alpha, print, graph
<T>	<t>	upper, alpha, print, graph
<U>	<u>	upper, alpha, print, graph
<V>	<v>	upper, alpha, print, graph
<W>	<w>	upper, alpha, print, graph
<X>	<x>	upper, alpha, print, graph
<Y>	<y>	upper, alpha, print, graph
<Z>	<z>	upper, alpha, print, graph
<left-square-bracket>		punct, print, graph
<backslash>		punct, print, graph
<right-square-bracket>		punct, print, graph

4373
4374
4375
4376
4377
4378
4379
4380
4381
4382
4383
4384
4385
4386
4387
4388
4389
4390
4391
4392
4393
4394
4395
4396
4397
4398
4399
4400
4401
4402
4403
4404
4405
4406
4407
4408

Symbolic Name	Other Case	Character Classes
<circumflex>		punct, print, graph
<underscore>		punct, print, graph
<grave-accent>		punct, print, graph
<a>	<A>	lower, xdigit, alpha, print, graph
		lower, xdigit, alpha, print, graph
<c>	<C>	lower, xdigit, alpha, print, graph
<d>	<D>	lower, xdigit, alpha, print, graph
<e>	<E>	lower, xdigit, alpha, print, graph
<f>	<F>	lower, xdigit, alpha, print, graph
<g>	<G>	lower, alpha, print, graph
<h>	<H>	lower, alpha, print, graph
<i>	<I>	lower, alpha, print, graph
<j>	<J>	lower, alpha, print, graph
<k>	<K>	lower, alpha, print, graph
<l>	<L>	lower, alpha, print, graph
<m>	<M>	lower, alpha, print, graph
<n>	<N>	lower, alpha, print, graph
<o>	<O>	lower, alpha, print, graph
<p>	<P>	lower, alpha, print, graph
<q>	<Q>	lower, alpha, print, graph
<r>	<R>	lower, alpha, print, graph
<s>	<S>	lower, alpha, print, graph
<t>	<T>	lower, alpha, print, graph
<u>	<U>	lower, alpha, print, graph
<v>	<V>	lower, alpha, print, graph
<w>	<W>	lower, alpha, print, graph
<x>	<X>	lower, alpha, print, graph
<y>	<Y>	lower, alpha, print, graph
<z>	<Z>	lower, alpha, print, graph
<left-curly-bracket>		punct, print, graph
<vertical-line>		punct, print, graph
<right-curly-bracket>		punct, print, graph
<tilde>		punct, print, graph
		cntrl

4409 **7.3.2 LC_COLLATE**

4410 The *LC_COLLATE* category provides a collation sequence definition for numerous utilities in the
4411 Shell and Utilities volume of IEEE Std 1003.1-200x (*sort*, *uniq*, and so on), regular expression
4412 matching (see Chapter 9 (on page 165)) and the *strcoll()*, *strxfrm()*, *wscoll()*, and *wcsxfrm()*
4413 functions in the System Interfaces volume of IEEE Std 1003.1-200x.

4414 A collation sequence definition shall define the relative order between collating elements
4415 (characters and multi-character collating elements) in the locale. This order is expressed in terms
4416 of collation values; that is, by assigning each element one or more collation values (also known
4417 as collation weights). This does not imply that implementations shall assign such values, but
4418 that ordering of strings using the resultant collation definition in the locale behaves as if such
4419 assignment is done and used in the collation process. At least the following capabilities are
4420 provided:

- 4421 1. **Multi-character collating elements.** Specification of multi-character collating elements
4422 (that is, sequences of two or more characters to be collated as an entity).

- 4423 2. **User-defined ordering of collating elements.** Each collating element shall be assigned a
 4424 collation value defining its order in the character (or basic) collation sequence. This
 4425 ordering is used by regular expressions and pattern matching and, unless collation weights
 4426 are explicitly specified, also as the collation weight to be used in sorting.
- 4427 3. **Multiple weights and equivalence classes.** Collating elements can be assigned one or
 4428 more (up to the limit {COLL_WEIGHTS_MAX}, as defined in <limits.h>) collating weights
 4429 for use in sorting. The first weight is hereafter referred to as the primary weight.
- 4430 4. **One-to-many mapping.** A single character is mapped into a string of collating elements.
- 4431 5. **Equivalence class definition.** Two or more collating elements have the same collation
 4432 value (primary weight).
- 4433 6. **Ordering by weights.** When two strings are compared to determine their relative order,
 4434 the two strings are first broken up into a series of collating elements; the elements in each
 4435 successive pair of elements are then compared according to the relative primary weights
 4436 for the elements. If equal, and more than one weight has been assigned, then the pairs of
 4437 collating elements are recompared according to the relative subsequent weights, until
 4438 either a pair of collating elements compare unequal or the weights are exhausted.

4439 The following keywords shall be recognized in a collation sequence definition. They are
 4440 described in detail in the following sections.

4441	copy	Specify the name of an existing locale which shall be used as the	
4442		definition of this category. If this keyword is specified, no other keyword	
4443		shall be specified.	
4444	collating-element	Define a collating-element symbol representing a multi-character	
4445		collating element. This keyword is optional.	
4446	collating-symbol	Define a collating symbol for use in collation order statements. This	
4447		keyword is optional.	
4448	order_start	Define collation rules. This statement shall be followed by one or more	
4449		collation order statements, assigning character collation values and	
4450		collation weights to collating elements.	
4451	order_end	Specify the end of the collation-order statements.	

4452 7.3.2.1 *The collating-element Keyword*

4453 In addition to the collating elements in the character set, the **collating-element** keyword can be
 4454 used to define multi-character collating elements. The syntax is as follows:

```
4455     "collating-element %s from \"%s\"\\n", <collating-symbol>, <string>
```

4456 The <collating-symbol> operand shall be a symbolic name, enclosed between angle brackets ('<' and '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. The string operand is a string of two or more characters that collates as an entity. A <collating-element> defined via this keyword is only recognized with the *LC_COLLATE* category.

4461 For example:

```
4462     collating-element <ch> from "<c><h>"
4463     collating-element <e-acute> from "<acute><e>"
4464     collating-element <ll> from "ll"
```

4465 7.3.2.2 *The collating-symbol Keyword*

4466 This keyword shall be used to define symbols for use in collation sequence statements; that is,
4467 between the **order_start** and the **order_end** keywords. The syntax is as follows:

4468 "collating-symbol %s\n", <collating-symbol>

4469 The <collating-symbol> shall be a symbolic name, enclosed between angle brackets ('<' and
4470 '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any
4471 other symbolic name defined in this collation definition. A <collating-symbol> defined via this
4472 keyword is only recognized within the *LC_COLLATE* category.

4473 For example:

4474 collating-symbol <UPPER_CASE>
4475 collating-symbol <HIGH>

4476 The **collating-symbol** keyword defines a symbolic name that can be associated with a relative
4477 position in the character order sequence. While such a symbolic name does not represent any
4478 collating element, it can be used as a weight.

4479 7.3.2.3 *The order_start Keyword*

4480 The **order_start** keyword shall precede collation order entries and also define the number of
4481 weights for this collation sequence definition and other collation rules. The syntax is as follows:

4482 "order_start %s;%s;...;%s\n", <sort-rules>, <sort-rules> ...

4483 The operands to the **order_start** keyword are optional. If present, the operands define rules to be
4484 applied when strings are compared. The number of operands define how many weights each
4485 element is assigned; if no operands are present, one **forward** operand is assumed. If present, the
4486 first operand defines rules to be applied when comparing strings using the first (primary)
4487 weight; the second when comparing strings using the second weight, and so on. Operands shall
4488 be separated by semicolons (';'). Each operand shall consist of one or more collation
4489 directives, separated by commas (','). If the number of operands exceeds the
4490 {COLL_WEIGHTS_MAX} limit, the utility shall issue a warning message. The following
4491 directives shall be supported:

4492 **forward** Specifies that comparison operations for the weight level shall proceed from start
4493 of string towards the end of string.

4494 **backward** Specifies that comparison operations for the weight level shall proceed from end of
4495 string towards the beginning of string.

4496 **position** Specifies that comparison operations for the weight level shall consider the relative
4497 position of elements in the strings not subject to **IGNORE**. The string containing
4498 an element not subject to **IGNORE** after the fewest collating elements subject to
4499 **IGNORE** from the start of the compare shall collate first. If both strings contain a
4500 character not subject to **IGNORE** in the same relative position, the collating values
4501 assigned to the elements shall determine the ordering. In case of equality,
4502 subsequent characters not subject to **IGNORE** shall be considered in the same
4503 manner.

4504 The directives **forward** and **backward** are mutually-exclusive.

4505 If no operands are specified, a single **forward** operand shall be assumed.

4506 For example:

4507 order_start forward;backward

4508 7.3.2.4 Collation Order

4509 The **order_start** keyword shall be followed by collating identifier entries. The syntax for the
4510 collating element entries is as follows:

4511 "%s %s;%s;...;%s\n", <collating-identifier>, <weight>, <weight>, ...

4512 Each *collating-identifier* shall consist of either a character (in any of the forms defined in Section
4513 7.3 (on page 120)), a <collating-element>, a <collating-symbol>, an ellipsis, or the special symbol
4514 **UNDEFINED**. The order in which collating elements are specified determines the character
4515 order sequence, such that each collating element shall compare less than the elements following
4516 it.

4517 A <collating-element> shall be used to specify multi-character collating elements, and indicates
4518 that the character sequence specified via the <collating-element> is to be collated as a unit and in
4519 the relative order specified by its place.

4520 A <collating-symbol> can be used to define a position in the relative order for use in weights. No
4521 weights shall be specified with a <collating-symbol>.

4522 The ellipsis symbol specifies that a sequence of characters shall collate according to their
4523 encoded character values. It shall be interpreted as indicating that all characters with a coded
4524 character set value higher than the value of the character in the preceding line, and lower than
4525 the coded character set value for the character in the following line, in the current coded
4526 character set, shall be placed in the character collation order between the previous and the
4527 following character in ascending order according to their coded character set values. An initial
4528 ellipsis shall be interpreted as if the preceding line specified the NUL character, and a trailing
4529 ellipsis as if the following line specified the highest coded character set value in the current
4530 coded character set. An ellipsis shall be treated as invalid if the preceding or following lines do
4531 not specify characters in the current coded character set. The use of the ellipsis symbol ties the
4532 definition to a specific coded character set and may preclude the definition from being portable
4533 between implementations.

4534 The symbol **UNDEFINED** shall be interpreted as including all coded character set values not
4535 specified explicitly or via the ellipsis symbol. Such characters shall be inserted in the character
4536 collation order at the point indicated by the symbol, and in ascending order according to their
4537 coded character set values. If no **UNDEFINED** symbol is specified, and the current coded
4538 character set contains characters not specified in this section, the utility shall issue a warning
4539 message and place such characters at the end of the character collation order.

4540 The optional operands for each collation-element shall be used to define the primary, secondary,
4541 or subsequent weights for the collating element. The first operand specifies the relative primary
4542 weight, the second the relative secondary weight, and so on. Two or more collation-elements can
4543 be assigned the same weight; they belong to the same *equivalence class* if they have the same
4544 primary weight. Collation shall behave as if, for each weight level, elements subject to **IGNORE**
4545 are removed, unless the **position** collation directive is specified for the corresponding level with
4546 the **order_start** keyword. Then each successive pair of elements shall be compared according to
4547 the relative weights for the elements. If the two strings compare equal, the process shall be
4548 repeated for the next weight level, up to the limit {COLL_WEIGHTS_MAX}.

4549 Weights shall be expressed as characters (in any of the forms specified in Section 7.3 (on page
4550 120)), <collating-symbol>s, <collating-element>s, an ellipsis, or the special symbol **IGNORE**. A
4551 single character, a <collating-symbol>, or a <collating-element> shall represent the relative position
4552 in the character collating sequence of the character or symbol, rather than the character or
4553 characters themselves. Thus, rather than assigning absolute values to weights, a particular

4554 weight is expressed using the relative order value assigned to a collating element based on its
 4555 order in the character collation sequence.

4556 One-to-many mapping is indicated by specifying two or more concatenated characters or
 4557 symbolic names. For example, if the <eszet> is given the string "<s><s>" as a weight,
 4558 comparisons are performed as if all occurrences of the <eszet> are replaced by "<s><s>"
 4559 (assuming that "<s>" has the collating weight "<s>"). If it is necessary to define <eszet> and
 4560 "<s><s>" as an equivalence class, then a collating element must be defined for the string "ss".

4561 All characters specified via an ellipsis shall by default be assigned unique weights, equal to the
 4562 relative order of characters. Characters specified via an explicit or implicit **UNDEFINED** special
 4563 symbol shall by default be assigned the same primary weight (that is, they belong to the same
 4564 equivalence class). An ellipsis symbol as a weight shall be interpreted to mean that each
 4565 character in the sequence shall have unique weights, equal to the relative order of their character
 4566 in the character collation sequence. The use of the ellipsis as a weight shall be treated as an error
 4567 if the collating element is neither an ellipsis nor the special symbol **UNDEFINED**.

4568 The special keyword **IGNORE** as a weight shall indicate that when strings are compared using
 4569 the weights at the level where **IGNORE** is specified, the collating element shall be ignored; that
 4570 is, as if the string did not contain the collating element. In regular expressions and pattern
 4571 matching, all characters that are subject to **IGNORE** in their primary weight form an
 4572 equivalence class.

4573 An empty operand shall be interpreted as the collating element itself.

4574 For example, the order statement:

```
4575     <a>     <a> ; <a>
```

4576 is equal to:

```
4577     <a>
```

4578 An ellipsis can be used as an operand if the collating element was an ellipsis, and shall be
 4579 interpreted as the value of each character defined by the ellipsis.

4580 The collation order as defined in this section affects the interpretation of bracket expressions in
 4581 regular expressions (see Section 9.3.5 (on page 168)).

4582 For example:

```
4583     order_start  forward ; backward
4584     UNDEFINED   IGNORE ; IGNORE
4585     <LOW>
4586     <space>     <LOW> ; <space>
4587     ...         <LOW> ; ...
4588     <a>         <a> ; <a>
4589     <a-acute>   <a> ; <a-acute>
4590     <a-grave>   <a> ; <a-grave>
4591     <A>        <a> ; <A>
4592     <A-acute>   <a> ; <A-acute>
4593     <A-grave>   <a> ; <A-grave>
4594     <ch>       <ch> ; <ch>
4595     <Ch>       <ch> ; <Ch>
4596     <s>        <s> ; <s>
4597     <eszet>    "<s><s>" ; "<eszet><eszet>"
4598     order_end
```

4599 This example is interpreted as follows:

- 4600 1. The **UNDEFINED** means that all characters not specified in this definition (explicitly or
4601 via the ellipsis) shall be ignored for collation purposes.
- 4602 2. All characters between <space> and 'a' shall have the same primary equivalence class
4603 and individual secondary weights based on their ordinal encoded values.
- 4604 3. All characters based on the uppercase or lowercase character 'a' belong to the same
4605 primary equivalence class.
- 4606 4. The multi-character collating element <ch> is represented by the collating symbol <ch>
4607 and belongs to the same primary equivalence class as the multi-character collating element
4608 <Ch>.

4609 7.3.2.5 *The order_end Keyword*

4610 The collating order entries shall be terminated with an **order_end** keyword.

4611 7.3.2.6 *LC_COLLATE Category in the POSIX Locale*

4612 The collation sequence definition of the POSIX locale follows; the code listing depicts the
4613 *localedef* input.

```
4614 LC_COLLATE
4615 # This is the POSIX locale definition for the LC_COLLATE category.
4616 # The order is the same as in the ASCII codeset.
4617 order_start forward
4618 <NUL>
4619 <SOH>
4620 <STX>
4621 <ETX>
4622 <EOT>
4623 <ENQ>
4624 <ACK>
4625 <alert>
4626 <backspace>
4627 <tab>
4628 <newline>
4629 <vertical-tab>
4630 <form-feed>
4631 <carriage-return>
4632 <SO>
4633 <SI>
4634 <DLE>
4635 <DC1>
4636 <DC2>
4637 <DC3>
4638 <DC4>
4639 <NAK>
4640 <SYN>
4641 <ETB>
4642 <CAN>
4643 <EM>
4644 <SUB>
4645 <ESC>
```

4646 <IS4>
4647 <IS3>
4648 <IS2>
4649 <IS1>
4650 <space>
4651 <exclamation-mark>
4652 <quotation-mark>
4653 <number-sign>
4654 <dollar-sign>
4655 <percent-sign>
4656 <ampersand>
4657 <apostrophe>
4658 <left-parenthesis>
4659 <right-parenthesis>
4660 <asterisk>
4661 <plus-sign>
4662 <comma>
4663 <hyphen>
4664 <period>
4665 <slash>
4666 <zero>
4667 <one>
4668 <two>
4669 <three>
4670 <four>
4671 <five>
4672 <six>
4673 <seven>
4674 <eight>
4675 <nine>
4676 <colon>
4677 <semicolon>
4678 <less-than-sign>
4679 <equals-sign>
4680 <greater-than-sign>
4681 <question-mark>
4682 <commercial-at>
4683 <A>
4684
4685 <C>
4686 <D>
4687 <E>
4688 <F>
4689 <G>
4690 <H>
4691 <I>
4692 <J>
4693 <K>
4694 <L>
4695 <M>
4696 <N>
4697 <O>

```

4698      <P>
4699      <Q>
4700      <R>
4701      <S>
4702      <T>
4703      <U>
4704      <V>
4705      <W>
4706      <X>
4707      <Y>
4708      <Z>
4709      <left-square-bracket>
4710      <backslash>
4711      <right-square-bracket>
4712      <circumflex>
4713      <underscore>
4714      <grave-accent>
4715      <a>
4716      <b>
4717      <c>
4718      <d>
4719      <e>
4720      <f>
4721      <g>
4722      <h>
4723      <i>
4724      <j>
4725      <k>
4726      <l>
4727      <m>
4728      <n>
4729      <o>
4730      <p>
4731      <q>
4732      <r>
4733      <s>
4734      <t>
4735      <u>
4736      <v>
4737      <w>
4738      <x>
4739      <y>
4740      <z>
4741      <left-curly-bracket>
4742      <vertical-line>
4743      <right-curly-bracket>
4744      <tilde>
4745      <DEL>
4746      order_end
4747      #
4748      END LC_COLLATE

```

4749 **7.3.3 LC_MONETARY**

4750 The *LC_MONETARY* category shall define the rules and symbols that are used to format
 4751 XSI monetary numeric information. This information is available through the *localeconv()* function
 4752 and is used by the *strfmon()* function.

4753 XSI Some of the information is also available in an alternative form via the *nl_langinfo()* function
 4754 (see CRNCYSTR in *<langinfo.h>*).

4755 The following items are defined in this category of the locale. The item names are the keywords
 4756 recognized by the *localedef* utility when defining a locale. They are also similar to the member
 4757 names of the *lconv* structure defined in *<locale.h>*; see *<locale.h>* for the exact symbols in the
 4758 header. The *localeconv()* function returns {CHAR_MAX} for unspecified integer items and the
 4759 empty string (" ") for unspecified or size zero string items.

4760 In a locale definition file, the operands are strings, formatted as indicated by the grammar in
 4761 Section 7.4 (on page 149). For some keywords, the strings can contain only integers. Keywords
 4762 that are not provided, string values set to the empty string (" "), or integer keywords set to -1,
 4763 are used to indicate that the value is not available in the locale. The following keywords shall be
 4764 recognized:

4765 **copy** Specify the name of an existing locale which shall be used as the |
 4766 definition of this category. If this keyword is specified, no other keyword |
 4767 shall be specified. |

4768 **Note:** This is a *localedef* utility keyword, unavailable through *localeconv()*.

4769 **int_curr_symbol** The international currency symbol. The operand shall be a four-character |
 4770 string, with the first three characters containing the alphabetic |
 4771 international currency symbol in accordance with those specified in the |
 4772 ISO 4217: 1995 standard. The fourth character shall be the character used |
 4773 to separate the international currency symbol from the monetary |
 4774 quantity.

4775 **currency_symbol** The string that shall be used as the local currency symbol.

4776 **mon_decimal_point** The operand is a string containing the symbol that shall be used as the |
 4777 decimal delimiter (radix character) in monetary formatted quantities. |

4778 **mon_thousands_sep** The operand is a string containing the symbol that shall be used as a |
 4779 separator for groups of digits to the left of the decimal delimiter in |
 4780 formatted monetary quantities. |

4781 **mon_grouping** Define the size of each group of digits in formatted monetary quantities.
 4782 The operand is a sequence of integers separated by semicolons. Each
 4783 integer specifies the number of digits in each group, with the initial
 4784 integer defining the size of the group immediately preceding the decimal
 4785 delimiter, and the following integers defining the preceding groups. If the
 4786 last integer is not -1, then the size of the previous group (if any) shall be
 4787 repeatedly used for the remainder of the digits. If the last integer is -1,
 4788 then no further grouping shall be performed.

4789 **positive_sign** A string that shall be used to indicate a non-negative-valued formatted
 4790 monetary quantity.

4791 **negative_sign** A string that shall be used to indicate a negative-valued formatted
 4792 monetary quantity.

4793 **int_frac_digits** An integer representing the number of fractional digits (those to the right
 4794 of the decimal delimiter) to be written in a formatted monetary quantity

4795		using int_curr_symbol .
4796	frac_digits	An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using currency_symbol .
4797		
4798		
4799	p_cs_precedes	An integer set to 1 if the currency_symbol precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.
4800		
4801		
4802	p_sep_by_space	An integer set to 0 if no space separates the currency_symbol from the value for a monetary quantity with a non-negative value, set to 1 if a space separates the symbol from the value, and set to 2 if a space separates the symbol and the sign string, if adjacent.
4803		
4804		
4805		
4806	n_cs_precedes	An integer set to 1 if the currency_symbol precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.
4807		
4808		
4809	n_sep_by_space	An integer set to 0 if no space separates the currency_symbol from the value for a monetary quantity with a negative value, set to 1 if a space separates the symbol from the value, and set to 2 if a space separates the symbol and the sign string, if adjacent.
4810		
4811		
4812		
4813	p_sign_posn	An integer set to a value indicating the positioning of the positive_sign for a monetary quantity with a non-negative value. The following integer values shall be recognized for int_n_sign_posn , int_p_sign_posn , n_sign_posn , and p_sign_posn :
4814		
4815		
4816		
4817		0 Parentheses enclose the quantity and the currency_symbol .
4818		1 The sign string precedes the quantity and the currency_symbol .
4819		2 The sign string succeeds the quantity and the currency_symbol .
4820		3 The sign string precedes the currency_symbol .
4821		4 The sign string succeeds the currency_symbol .
4822	n_sign_posn	An integer set to a value indicating the positioning of the negative_sign for a negative formatted monetary quantity.
4823		
4824	int_p_cs_precedes	An integer set to 1 if the int_curr_symbol precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.
4825		
4826		
4827	int_n_cs_precedes	An integer set to 1 if the int_curr_symbol precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.
4828		
4829		
4830	int_p_sep_by_space	An integer to set 0 if no space separates the int_curr_symbol from the value for a monetary quantity with a non-negative value, set to 1 if a space separates the symbol from the value, and set to 2 if a space separates the symbol and the sign string, if adjacent.
4831		
4832		
4833		
4834	int_n_sep_by_space	An integer set to 0 if no space separates the int_curr_symbol from the value for a monetary quantity with a negative value, set to 1 if a space separates the symbol from the value, and set to 2 if a space separates the symbol and the sign string, if adjacent.
4835		
4836		
4837		

4886 **7.3.4 LC_NUMERIC**

4887 The *LC_NUMERIC* category shall define the rules and symbols that are used to format non-
 4888 XSI monetary numeric information. This information is available through the *localeconv()* function.
 4889 Some of the information is also available in an alternative form via the *nl_langinfo()* function.

4890 The following items are defined in this category of the locale. The item names are the keywords
 4891 recognized by the *localedef* utility when defining a locale. They are also similar to the member
 4892 names of the *lconv* structure defined in `<locale.h>`; see `<locale.h>` for the exact symbols in the
 4893 header. The *localeconv()* function returns `{CHAR_MAX}` for unspecified integer items and the
 4894 empty string (" ") for unspecified or size zero string items.

4895 In a locale definition file, the operands are strings, formatted as indicated by the grammar in
 4896 Section 7.4 (on page 149). For some keywords, the strings can only contain integers. Keywords
 4897 that are not provided, string values set to the empty string (" "), or integer keywords set to `-1`,
 4898 shall be used to indicate that the value is not available in the locale. The following keywords
 4899 shall be recognized:

4900 **copy** Specify the name of an existing locale which shall be used as the definition of |
 4901 this category. If this keyword is specified, no other keyword shall be specified. |

4902 **Note:** This is a *localedef* utility keyword, unavailable through *localeconv()*.

4903 **decimal_point** The operand is a string containing the symbol that shall be used as the
 4904 decimal delimiter (radix character) in numeric, non-monetary formatted
 4905 quantities. This keyword cannot be omitted and cannot be set to the empty
 4906 string. In contexts where standards limit the **decimal_point** to a single byte,
 4907 the result of specifying a multi-byte operand shall be unspecified.

4908 **thousands_sep** The operand is a string containing the symbol that shall be used as a separator
 4909 for groups of digits to the left of the decimal delimiter in numeric, non-
 4910 monetary formatted monetary quantities. In contexts where standards limit
 4911 the **thousands_sep** to a single byte, the result of specifying a multi-byte
 4912 operand shall be unspecified.

4913 **grouping** Define the size of each group of digits in formatted non-monetary quantities.
 4914 The operand is a sequence of integers separated by semicolons. Each integer
 4915 specifies the number of digits in each group, with the initial integer defining
 4916 the size of the group immediately preceding the decimal delimiter, and the
 4917 following integers defining the preceding groups. If the last integer is not `-1`,
 4918 then the size of the previous group (if any) shall be repeatedly used for the
 4919 remainder of the digits. If the last integer is `-1`, then no further grouping shall
 4920 be performed.

4921 **7.3.4.1 LC_NUMERIC Category in the POSIX Locale**

4922 The non-monetary numeric formatting definitions for the POSIX locale follow; the code listing
 4923 depicting the *localedef* input, the table representing the same information with the addition of
 4924 XSI *localeconv()* values, and *nl_langinfo()* constants.

```
4925 LC_NUMERIC
4926 # This is the POSIX locale definition for
4927 # the LC_NUMERIC category.
4928 #
4929 decimal_point    "<period>"
4930 thousands_sep    " "
4931 grouping         -1
4932 #
```

4933 END LC_NUMERIC

Item	langinfo Constant	POSIX Locale Value	localeconv() Value	localedef Value
decimal_point	RADIXCHAR	."	."	.
thousands_sep	THOUSEP	N/A	."	."
grouping	—	N/A	."	-1

4939 XSI In the preceding table, the **langinfo Constant** column represents an XSI-conforming extension.
 4940 The entry N/A indicates that the value is not available in the POSIX locale.

4941 **7.3.5 LC_TIME**

4942 The *LC_TIME* category shall define the interpretation of the conversion specifications supported
 4943 XSI by the *date* utility and shall affect the behavior of the *strftime()*, *wcsftime()*, *strptime()*, and
 4944 *nl_langinfo()* functions. Since the interfaces for C-language access and locale definition differ
 4945 significantly, they are described separately.

4946 **7.3.5.1 LC_TIME Locale Definition**

4947 For locale definition, the following mandatory keywords shall be recognized:

4948 **copy** Specify the name of an existing locale which shall be used as the definition of
 4949 this category. If this keyword is specified, no other keyword shall be specified.

4950 **abday** Define the abbreviated weekday names, corresponding to the %a conversion
 4951 specification (conversion specification in the *strptime()*, *wcsftime()*, and
 4952 *strptime()* functions). The operand shall consist of seven semicolon-separated
 4953 strings, each surrounded by double-quotes. The first string shall be the
 4954 abbreviated name of the day corresponding to Sunday, the second the
 4955 abbreviated name of the day corresponding to Monday, and so on.

4956 **day** Define the full weekday names, corresponding to the %A conversion
 4957 specification. The operand shall consist of seven semicolon-separated strings,
 4958 each surrounded by double-quotes. The first string is the full name of the day
 4959 corresponding to Sunday, the second the full name of the day corresponding
 4960 to Monday, and so on.

4961 **abmon** Define the abbreviated month names, corresponding to the %b conversion
 4962 specification. The operand shall consist of twelve semicolon-separated strings,
 4963 each surrounded by double-quotes. The first string shall be the abbreviated
 4964 name of the first month of the year (January), the second the abbreviated
 4965 name of the second month, and so on.

4966 **mon** Define the full month names, corresponding to the %B conversion
 4967 specification. The operand shall consist of twelve semicolon-separated strings,
 4968 each surrounded by double-quotes. The first string shall be the full name of
 4969 the first month of the year (January), the second the full name of the second
 4970 month, and so on.

4971 **d_t_fmt** Define the appropriate date and time representation, corresponding to the %c
 4972 conversion specification. The operand shall consist of a string containing any
 4973 combination of characters and conversion specifications. In addition, the
 4974 string can contain escape sequences defined in the table in Table 5-1 (on page
 4975 108) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v').

4976	d_fmt	Define the appropriate date representation, corresponding to the %X conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in the table in Table 5-1 (on page 108).
4977		
4978		
4979		
4980		
4981	t_fmt	Define the appropriate time representation, corresponding to the %X conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in the table in Table 5-1 (on page 108).
4982		
4983		
4984		
4985		
4986	am_pm	Define the appropriate representation of the <i>ante meridiem</i> and <i>post meridiem</i> strings, corresponding to the %P conversion specification. The operand shall consist of two strings, separated by a semicolon, each surrounded by double-quotes. The first string shall represent the <i>ante meridiem</i> designation, the last string the <i>post meridiem</i> designation.
4987		
4988		
4989		
4990		
4991	t_fmt_ampm	Define the appropriate time representation in the 12-hour clock format with am_pm , corresponding to the %r conversion specification. The operand shall consist of a string and can contain any combination of characters and conversion specifications. If the string is empty, the 12-hour format is not supported in the locale.
4992		
4993		
4994		
4995		
4996	era	Define how years are counted and displayed for each era in a locale. The operand shall consist of semicolon-separated strings. Each string shall be an era description segment with the format: <i>direction:offset:start_date:end_date:era_name:era_format</i> according to the definitions below. There can be as many era description segments as are necessary to describe the different eras.
4997		
4998		
4999		
5000		
5001		
5002	Note:	The start of an era might not be the earliest point in the era—it may be the latest. For example, the Christian era BC starts on the day before January 1, AD 1, and increases with earlier time.
5003		
5004		
5005	<i>direction</i>	Either a '+' or a '-' character. The '+' character shall indicate that years closer to the <i>start_date</i> have lower numbers than those closer to the <i>end_date</i> . The '-' character shall indicate that years closer to the <i>start_date</i> have higher numbers than those closer to the <i>end_date</i> .
5006		
5007		
5008		
5009		
5010	<i>offset</i>	The number of the year closest to the <i>start_date</i> in the era, corresponding to the %EY conversion specification.
5011		
5012	<i>start_date</i>	A date in the form yyyy/mm/dd, where yyyy, mm, and dd are the year, month, and day numbers respectively of the start of the era. Years prior to AD 1 shall be represented as negative numbers.
5013		
5014		
5015		
5016	<i>end_date</i>	The ending date of the era, in the same format as the <i>start_date</i> , or one of the two special values "-*" or "+*". The value "-*" shall indicate that the ending date is the beginning of time. The value "+*" shall indicate that the ending date is the end of time.
5017		
5018		
5019		
5020	<i>era_name</i>	A string representing the name of the era, corresponding to the %EC conversion specification.
5021		

5022		<i>era_format</i>	A string for formatting the year in the era, corresponding to the
5023			%EY conversion specification.
5024	era_d_fmt		Define the format of the date in alternative era notation, corresponding to the
5025			%Ex conversion specification.
5026	era_t_fmt		Define the locale's appropriate alternative time format, corresponding to the
5027			%EX conversion specification.
5028	era_d_t_fmt		Define the locale's appropriate alternative date and time format,
5029			corresponding to the %Ec conversion specification.
5030	alt_digits		Define alternative symbols for digits, corresponding to the %O modified
5031			conversion specification. The operand shall consist of semicolon-separated
5032			strings, each surrounded by double-quotes. The first string shall be the
5033			alternative symbol corresponding with zero, the second string the symbol
5034			corresponding with one, and so on. Up to 100 alternative symbol strings can
5035			be specified. The %O modifier shall indicate that the string corresponding to
5036			the value specified via the conversion specification shall be used instead of the
5037			value.
5038	7.3.5.2	<i>LC_TIME C-Language Access</i>	
5039	XSI		This section describes extensions to access information in the <i>LC_TIME</i> category using the
5040			<i>nl_langinfo()</i> function. This functionality is dependent on support of the XSI extension (and the
5041			rest of this section is not further shaded for this option).
5042			The following constants used to identify items of <i>langinfo</i> data can be used as arguments to the
5043			<i>nl_langinfo()</i> function to access information in the <i>LC_TIME</i> category. These constants are
5044			defined in the < langinfo.h > header.
5045	ABDAY_x		The abbreviated weekday names (for example Sun), where <i>x</i> is a number from
5046			1 to 7.
5047	DAY_x		The full weekday names (for example Sunday), where <i>x</i> is a number from 1 to
5048			7.
5049	ABMON_x		The abbreviated month names (for example Jan), where <i>x</i> is a number from 1
5050			to 12.
5051	MON_x		The full month names (for example January), where <i>x</i> is a number from 1 to
5052			12.
5053	D_T_FMT		The appropriate date and time representation.
5054	D_FMT		The appropriate date representation.
5055	T_FMT		The appropriate time representation.
5056	AM_STR		The appropriate ante-meridiem affix.
5057	PM_STR		The appropriate post-meridiem affix.
5058	T_FMT_AMPM		The appropriate time representation in the 12-hour clock format with
5059			AM_STR and PM_STR.
5060	ERA		The era description segments, which describe how years are counted and
5061			displayed for each era in a locale. Each era description segment shall have the
5062			format:

```

5063         direction:offset:start_date:end_date:era_name:era_format
5064         according to the definitions below. There can be as many era description
5065         segments as are necessary to describe the different eras. Era description
5066         segments are separated by semicolons.
5067         direction      Either a '+' or a '-' character. The '+' character shall indicate
5068         that years closer to the start_date have lower numbers than those
5069         closer to the end_date. The '-' character shall indicate that
5070         years closer to the start_date have higher numbers than those
5071         closer to the end_date.
5072         offset         The number of the year closest to the start_date in the era.
5073         start_date      A date in the form yyyy/mm/dd, where yyyy, mm, and dd are the
5074         year, month, and day numbers respectively of the start of the
5075         era. Years prior to AD 1 shall be represented as negative
5076         numbers.
5077         end_date        The ending date of the era, in the same format as the start_date,
5078         or one of the two special values "-*" or "+*". The value "-*"
5079         shall indicate that the ending date is the beginning of time. The
5080         value "+*" shall indicate that the ending date is the end of time.
5081         era_name         The era, corresponding to the %EC conversion specification.
5082         era_format      The format of the year in the era, corresponding to the %EY
5083         conversion specification.
5084         ERA_D_FMT        The era date format.
5085         ERA_T_FMT        The locale's appropriate alternative time format, corresponding to the %EX
5086         conversion specification.
5087         ERA_D_T_FMT      The locale's appropriate alternative date and time format, corresponding to
5088         the %Ec conversion specification.
5089         ALT_DIGITS       The alternative symbols for digits, corresponding to the %O conversion
5090         specification modifier. The value consists of semicolon-separated symbols.
5091         The first is the alternative symbol corresponding to zero, the second is the
5092         symbol corresponding to one, and so on. Up to 100 alternative symbols may
5093         be specified.
5094 7.3.5.3 LC_TIME Category in the POSIX Locale
5095         The LC_TIME category definition of the POSIX locale follows; the code listing depicts the
5096         localedef input; the table represents the same information with the addition of localedef keywords,
5097         conversion specifiers used by the date utility and the strptime(), wcsftime(), and strptime()
5098         XSI functions, and nl_langinfo() constants.
5099         LC_TIME
5100         # This is the POSIX locale definition for
5101         # the LC_TIME category.
5102         #
5103         # Abbreviated weekday names (%a)
5104         abday      "<S><u><n>" ; "<M><o><n>" ; "<T><u><e>" ; "<W><e><d>" ; \
5105                 "<T><h><u>" ; "<F><r><i>" ; "<S><a><t>"
5106         #
5107         # Full weekday names (%A)

```

```

5108     day          "<S><u><n><d><a><y>" ; "<M><o><n><d><a><y>" ; \
5109             "<T><u><e><s><d><a><y>" ; "<W><e><d><n><e><s><d><a><y>" ; \
5110             "<T><h><u><r><s><d><a><y>" ; "<F><r><i><d><a><y>" ; \
5111             "<S><a><t><u><r><d><a><y>"
5112     #
5113     # Abbreviated month names (%b)
5114     abmon         "<J><a><n>" ; "<F><e><b>" ; "<M><a><r>" ; \
5115             "<A><p><r>" ; "<M><a><y>" ; "<J><u><n>" ; \
5116             "<J><u><l>" ; "<A><u><g>" ; "<S><e><p>" ; \
5117             "<O><c><t>" ; "<N><o><v>" ; "<D><e><c>"
5118     #
5119     # Full month names (%B)
5120     mon           "<J><a><n><u><a><r><y>" ; "<F><e><b><r><u><a><r><y>" ; \
5121             "<M><a><r><c><h>" ; "<A><p><r><i><l>" ; \
5122             "<M><a><y>" ; "<J><u><n><e>" ; \
5123             "<J><u><l><y>" ; "<A><u><g><u><s><t>" ; \
5124             "<S><e><p><t><e><m><b><e><r>" ; "<O><c><t><o><b><e><r>" ; \
5125             "<N><o><v><e><m><b><e><r>" ; "<D><e><c><e><m><b><e><r>"
5126     #
5127     # Equivalent of AM/PM (%p)          "AM" ; "PM"
5128     am_pm         "<A><M>" ; "<P><M>"
5129     #
5130     # Appropriate date and time representation (%c)
5131     #      "%a %b %e %H:%M:%S %Y"
5132     d_t_fmt       "<percent-sign><a><space><percent-sign><b>\
5133             <space><percent-sign><e><space><percent-sign><H>\
5134             <colon><percent-sign><M><colon><percent-sign><S>\
5135             <space><percent-sign><Y>"
5136     #
5137     # Appropriate date representation (%x)  "%m/%d/%y"
5138     d_fmt         "<percent-sign><m><slash><percent-sign><d>\
5139             <slash><percent-sign><y>"
5140     #
5141     # Appropriate time representation (%X)  "%H:%M:%S"
5142     t_fmt         "<percent-sign><H><colon><percent-sign><M>\
5143             <colon><percent-sign><S>"
5144     #
5145     # Appropriate 12-hour time representation (%r) "%I:%M:%S %p"
5146     t_fmt_ampm   "<percent-sign><I><colon><percent-sign><M><colon>\
5147             <percent-sign><S><space><percent_sign><p>"
5148     #
5149     END LC_TIME

```

5150
5151
5152
5153
5154
5155

localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
d_t_fmt	D_T_FMT	%c	"%a %b %e %H:%M:%S %Y"
d_fmt	D_FMT	%x	"%m/%d/%y"
t_fmt	T_FMT	%X	"%H:%M:%S"

5156
5157
5158
5159
5160
5161
5162
5163
5164
5165
5166
5167
5168
5169
5170
5171
5172
5173
5174
5175
5176
5177
5178
5179
5180
5181
5182
5183
5184
5185
5186
5187
5188
5189
5190
5191
5192
5193
5194
5195
5196
5197
5198
5199
5200
5201
5202
5203
5204

localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
am_pm	AM_STR	%p	"AM"
am_pm	PM_STR	%p	"PM"
t_fmt_ampm	T_FMT_AMPMPM	%r	"%I:%M:%S %p"
day	DAY_1	%A	"Sunday"
day	DAY_2	%A	"Monday"
day	DAY_3	%A	"Tuesday"
day	DAY_4	%A	"Wednesday"
day	DAY_5	%A	"Thursday"
day	DAY_6	%A	"Friday"
day	DAY_7	%A	"Saturday"
abday	ABDAY_1	%a	"Sun"
abday	ABDAY_2	%a	"Mon"
abday	ABDAY_3	%a	"Tue"
abday	ABDAY_4	%a	"Wed"
abday	ABDAY_5	%a	"Thu"
abday	ABDAY_6	%a	"Fri"
abday	ABDAY_7	%a	"Sat"
mon	MON_1	%B	"January"
mon	MON_2	%B	"February"
mon	MON_3	%B	"March"
mon	MON_4	%B	"April"
mon	MON_5	%B	"May"
mon	MON_6	%B	"June"
mon	MON_7	%B	"July"
mon	MON_8	%B	"August"
mon	MON_9	%B	"September"
mon	MON_10	%B	"October"
mon	MON_11	%B	"November"
mon	MON_12	%B	"December"
abmon	ABMON_1	%b	"Jan"
abmon	ABMON_2	%b	"Feb"
abmon	ABMON_3	%b	"Mar"
abmon	ABMON_4	%b	"Apr"
abmon	ABMON_5	%b	"May"
abmon	ABMON_6	%b	"Jun"
abmon	ABMON_7	%b	"Jul"
abmon	ABMON_8	%b	"Aug"
abmon	ABMON_9	%b	"Sep"
abmon	ABMON_10	%b	"Oct"
abmon	ABMON_11	%b	"Nov"
abmon	ABMON_12	%b	"Dec"
era	ERA	%EC, %Ey, %EY	N/A
era_d_fmt	ERA_D_FMT	%Ex	N/A
era_t_fmt	ERA_T_FMT	%EX	N/A
era_d_t_fmt	ERA_D_T_FMT	%Ec	N/A
alt_digits	ALT_DIGITS	%O	N/A

5205 XSI In the preceding table, the **langinfo Constant** column represents an XSI-conformant extension.

5206 The entry “N/A” indicates the value is not available in the POSIX locale.

5207 **7.3.6 LC_MESSAGES**

5208 The *LC_MESSAGES* category shall define the format and values used by various utilities for
 5209 XSI affirmative and negative responses. This information is available through the *nl_langinfo()*
 5210 function.

5211 XSI The message catalog used by the standard utilities and selected by the *catopen()* function shall be
 5212 determined by the setting of *NLSPATH*; see Chapter 8 (on page 157). The *LC_MESSAGES*
 5213 category can be specified as part of an *NLSPATH* substitution field.

5214 The following keywords shall be recognized as part of the locale definition file.

5215 **copy** Specify the name of an existing locale which shall be used as the definition of this |
 5216 category. If this keyword is specified, no other keyword shall be specified. |

5217 **Note:** This is a *localedef* keyword, unavailable through *nl_langinfo()*.

5218 **yesexpr** The operand consists of an extended regular expression (see Section 9.4 (on page
 5219 171)) that describes the acceptable affirmative response to a question expecting an
 5220 affirmative or negative response.

5221 **noexpr** The operand consists of an extended regular expression that describes the
 5222 acceptable negative response to a question expecting an affirmative or negative
 5223 response.

5224 **7.3.6.1 LC_MESSAGES Category for the POSIX Locale**

5225 The format and values for affirmative and negative responses of the POSIX locale follow; the
 5226 XSI code listing depicting the *localedef* input, the table representing the same information with the
 5227 addition of *nl_langinfo()* constants.

```
5228 LC_MESSAGES
5229 # This is the POSIX locale definition for
5230 # the LC_MESSAGES category.
5231 #
5232 yesexpr "<circumflex><left-square-bracket><y><Y><right-square-bracket>"
5233 #
5234 noexpr "<circumflex><left-square-bracket><n><N><right-square-bracket>"
5235 #
5236 END LC_MESSAGES
```

localedef Keyword	langinfo Constant	POSIX Locale Value
yesexpr	YEEXPR	"^[yY]"
noexpr	NOEXPR	"^[nN]"

5240 XSI In the preceding table, the **langinfo Constant** column represents an XSI-conformant extension.

5241 7.4 Locale Definition Grammar

5242 The grammar and lexical conventions in this section shall together describe the syntax for the
 5243 locale definition source. The general conventions for this style of grammar are described in the
 5244 Shell and Utilities volume of IEEE Std 1003.1-200x, Section 1.10, Grammar Conventions. The
 5245 grammar shall take precedence over the text in this chapter.

5246 7.4.1 Locale Lexical Conventions

5247 The lexical conventions for the locale definition grammar are described in this section.

5248 The following tokens shall be processed (in addition to those string constants shown in the
 5249 grammar):

5250	LOC_NAME	A string of characters representing the name of a locale.
5251	CHAR	Any single character.
5252	NUMBER	A decimal number, represented by one or more decimal digits.
5253	COLLSYMBOL	A symbolic name, enclosed between angle brackets. The string cannot duplicate any charmap symbol defined in the current charmap (if any), or a COLLELEMENT symbol.
5254		
5255		
5256	COLLELEMENT	A symbolic name, enclosed between angle brackets, which cannot duplicate either any charmap symbol or a COLLSYMBOL symbol.
5257		
5258	CHARCLASS	A string of alphanumeric characters from the portable character set, the first of which is not a digit, consisting of at least one and at most {CHARCLASS_NAME_MAX} bytes, and optionally surrounded by double-quotes.
5259		
5260		
5261		
5262	CHARSYMBOL	A symbolic name, enclosed between angle brackets, from the current charmap (if any).
5263		
5264	OCTAL_CHAR	One or more octal representations of the encoding of each byte in a single character. The octal representation consists of an escape character (normally a backslash) followed by two or more octal digits.
5265		
5266		
5267		
5268	HEX_CHAR	One or more hexadecimal representations of the encoding of each byte in a single character. The hexadecimal representation consists of an escape character followed by the constant x and two or more hexadecimal digits.
5269		
5270		
5271		
5272	DECIMAL_CHAR	One or more decimal representations of the encoding of each byte in a single character. The decimal representation consists of an escape character followed by a character 'd' and two or more decimal digits.
5273		
5274		
5275		
5276	ELLIPSIS	The string " . . . ".
5277	EXTENDED_REG_EXP	An extended regular expression as defined in the grammar in Section 9.5 (on page 175).
5278		
5279	EOL	The line termination character newline.

5280 **7.4.2 Locale Grammar**

5281 This section presents the grammar for the locale definition.

```

5282 %token          LOC_NAME
5283 %token          CHAR
5284 %token          NUMBER
5285 %token          COLLSYMBOL COLLELEMENT
5286 %token          CHARSYMBOL OCTAL_CHAR HEX_CHAR DECIMAL_CHAR
5287 %token          ELLIPSIS
5288 %token          EXTENDED_REG_EXP
5289 %token          EOL

5290 %start          locale_definition
5291 %%

5292 locale_definition : global_statements locale_categories
5293                  | locale_categories
5294                  ;

5295 global_statements : global_statements symbol_redefine
5296                  | symbol_redefine
5297                  ;

5298 symbol_redefine   : 'escape_char' CHAR EOL
5299                  | 'comment_char' CHAR EOL
5300                  ;

5301 locale_categories : locale_categories locale_category
5302                  | locale_category
5303                  ;

5304 locale_category   : lc_ctype | lc_collate | lc_messages
5305                  | lc_monetary | lc_numeric | lc_time
5306                  ;

5307 /* The following grammar rules are common to all categories */

5308 char_list         : char_list char_symbol
5309                  | char_symbol
5310                  ;

5311 char_symbol       : CHAR | CHARSYMBOL
5312                  | OCTAL_CHAR | HEX_CHAR | DECIMAL_CHAR
5313                  ;

5314 elem_list         : elem_list char_symbol
5315                  | elem_list COLLSYMBOL
5316                  | elem_list COLLELEMENT
5317                  | char_symbol
5318                  | COLLSYMBOL
5319                  | COLLELEMENT
5320                  ;

5321 symb_list         : symb_list COLLSYMBOL
5322                  | COLLSYMBOL
5323                  ;

```

```

5324     locale_name      : LOC_NAME
5325                       | ''' LOC_NAME '''
5326                       ;

5327     /* The following is the LC_CTYPE category grammar */

5328     lc_ctype          : ctype_hdr ctype_keywords      ctype_tlr
5329                       | ctype_hdr 'copy' locale_name EOL ctype_tlr
5330                       ;

5331     ctype_hdr         : 'LC_CTYPE' EOL
5332                       ;

5333     ctype_keywords    : ctype_keywords ctype_keyword
5334                       | ctype_keyword
5335                       ;

5336     ctype_keyword     : charclass_keyword charclass_list EOL
5337                       | charconv_keyword charconv_list EOL
5338                       | 'charclass' charclass_namelist EOL
5339                       ;

5340     charclass_namelist : charclass_namelist ';' CHARCLASS
5341                       | CHARCLASS
5342                       ;

5343     charclass_keyword : 'upper' | 'lower' | 'alpha' | 'digit'
5344                       | 'punct' | 'xdigit' | 'space' | 'print'
5345                       | 'graph' | 'blank' | 'cntrl' | 'alnum'
5346                       | CHARCLASS
5347                       ;

5348     charclass_list    : charclass_list ';' char_symbol
5349                       | charclass_list ';' ELLIPSIS ';' char_symbol
5350                       | char_symbol
5351                       ;

5352     charconv_keyword   : 'toupper'
5353                       | 'tolower'
5354                       ;

5355     charconv_list     : charconv_list ';' charconv_entry
5356                       | charconv_entry
5357                       ;

5358     charconv_entry    : '(' char_symbol ',' char_symbol ')'
5359                       ;

5360     ctype_tlr         : 'END' 'LC_CTYPE' EOL
5361                       ;

5362     /* The following is the LC_COLLATE category grammar */

5363     lc_collate        : collate_hdr collate_keywords    collate_tlr
5364                       | collate_hdr 'copy' locale_name EOL collate_tlr
5365                       ;

5366     collate_hdr       : 'LC_COLLATE' EOL
5367                       ;

```

```

5368     collate_keywords      :           order_statements
5369                               | opt_statements order_statements
5370                               ;

5371     opt_statements         : opt_statements collating_symbols
5372                               | opt_statements collating_elements
5373                               | collating_symbols
5374                               | collating_elements
5375                               ;

5376     collating_symbols      : 'collating-symbol' COLLSYMBOL EOL
5377                               ;

5378     collating_elements      : 'collating-element' COLLELEMENT
5379                               | 'from' ''' elem_list ''' EOL
5380                               ;

5381     order_statements        : order_start collation_order order_end
5382                               ;

5383     order_start             : 'order_start' EOL
5384                               | 'order_start' order_opts EOL
5385                               ;

5386     order_opts              : order_opts ';' order_opt
5387                               | order_opt
5388                               ;

5389     order_opt               : order_opt ',' opt_word
5390                               | opt_word
5391                               ;

5392     opt_word                 : 'forward' | 'backward' | 'position'
5393                               ;

5394     collation_order          : collation_order collation_entry
5395                               | collation_entry
5396                               ;

5397     collation_entry          : COLLSYMBOL EOL
5398                               | collation_element weight_list EOL
5399                               | collation_element                EOL
5400                               ;

5401     collation_element        : char_symbol
5402                               | COLLELEMENT
5403                               | ELLIPSIS
5404                               | 'UNDEFINED'
5405                               ;

5406     weight_list              : weight_list ';' weight_symbol
5407                               | weight_list ';'
5408                               | weight_symbol
5409                               ;

5410     weight_symbol            : /* empty */
5411                               | char_symbol
5412                               | COLLSYMBOL
5413                               | ''' elem_list '''

```

```

5414         | ''' symb_list '''
5415         | ELLIPSIS
5416         | 'IGNORE'
5417         ;
5418     order_end      : 'order_end' EOL
5419         ;
5420     collate_tlr    : 'END' 'LC_COLLATE' EOL
5421         ;
5422     /* The following is the LC_MESSAGES category grammar */
5423     lc_messages    : messages_hdr messages_keywords      messages_tlr
5424         | messages_hdr 'copy' locale_name EOL messages_tlr
5425         ;
5426     messages_hdr   : 'LC_MESSAGES' EOL
5427         ;
5428     messages_keywords : messages_keywords messages_keyword
5429         | messages_keyword
5430         ;
5431     messages_keyword : 'yesexpr' ''' EXTENDED_REG_EXP ''' EOL
5432         | 'noexpr'   ''' EXTENDED_REG_EXP ''' EOL
5433         ;
5434     messages_tlr   : 'END' 'LC_MESSAGES' EOL
5435         ;
5436     /* The following is the LC_MONETARY category grammar */
5437     lc_monetary    : monetary_hdr monetary_keywords      monetary_tlr
5438         | monetary_hdr 'copy' locale_name EOL monetary_tlr
5439         ;
5440     monetary_hdr   : 'LC_MONETARY' EOL
5441         ;
5442     monetary_keywords : monetary_keywords monetary_keyword
5443         | monetary_keyword
5444         ;
5445     monetary_keyword : mon_keyword_string mon_string EOL
5446         | mon_keyword_char NUMBER EOL
5447         | mon_keyword_char '-1' EOL
5448         | mon_keyword_grouping mon_group_list EOL
5449         ;
5450     mon_keyword_string : 'int_curr_symbol' | 'currency_symbol'
5451         | 'mon_decimal_point' | 'mon_thousands_sep'
5452         | 'positive_sign' | 'negative_sign'
5453         ;
5454     mon_string      : ''' char_list '''
5455         | '''
5456         ;

```

```

5457     mon_keyword_char      : 'int_frac_digits' | 'frac_digits'
5458                           | 'p_cs_precedes' | 'p_sep_by_space'
5459                           | 'n_cs_precedes' | 'n_sep_by_space'
5460                           | 'p_sign_posn' | 'n_sign_posn'
5461                           ;

5462     mon_keyword_grouping   : 'mon_grouping'
5463                           ;

5464     mon_group_list         : NUMBER
5465                           | mon_group_list ';' NUMBER
5466                           ;

5467     monetary_tlr          : 'END' 'LC_MONETARY' EOL
5468                           ;

5469     /* The following is the LC_NUMERIC category grammar */
5470     lc_numeric              : numeric_hdr numeric_keywords      numeric_tlr
5471                           | numeric_hdr 'copy' locale_name EOL numeric_tlr
5472                           ;

5473     numeric_hdr            : 'LC_NUMERIC' EOL
5474                           ;

5475     numeric_keywords       : numeric_keywords numeric_keyword
5476                           | numeric_keyword
5477                           ;

5478     numeric_keyword        : num_keyword_string num_string EOL
5479                           | num_keyword_grouping num_group_list EOL
5480                           ;

5481     num_keyword_string     : 'decimal_point'
5482                           | 'thousands_sep'
5483                           ;

5484     num_string              : '"' char_list '"'
5485                           | '""'
5486                           ;

5487     num_keyword_grouping   : 'grouping'
5488                           ;

5489     num_group_list         : NUMBER
5490                           | num_group_list ';' NUMBER
5491                           ;

5492     numeric_tlr            : 'END' 'LC_NUMERIC' EOL
5493                           ;

5494     /* The following is the LC_TIME category grammar */
5495     lc_time                 : time_hdr time_keywords           time_tlr
5496                           | time_hdr 'copy' locale_name EOL time_tlr
5497                           ;

5498     time_hdr                : 'LC_TIME' EOL
5499                           ;

```

```

5500     time_keywords      : time_keywords time_keyword
5501                          | time_keyword
5502                          ;
5503     time_keyword        : time_keyword_name time_list EOL
5504                          | time_keyword_fmt time_string EOL
5505                          | time_keyword_opt time_list EOL
5506                          ;
5507     time_keyword_name    : 'abday' | 'day' | 'abmon' | 'mon'
5508                          ;
5509     time_keyword_fmt     : 'd_t_fmt' | 'd_fmt' | 't_fmt'
5510                          | 'am_pm' | 't_fmt_ampm'
5511                          ;
5512     time_keyword_opt     : 'era' | 'era_d_fmt' | 'era_t_fmt'
5513                          | 'era_d_t_fmt' | 'alt_digits'
5514                          ;
5515     time_list            : time_list ';' time_string
5516                          | time_string
5517                          ;
5518     time_string          : '"' char_list '"'
5519                          ;
5520     time_tlr             : 'END' 'LC_TIME' EOL
5521                          ;

```


Environment Variables

5523

5524 8.1 Environment Variable Definition

5525 Environment variables defined in this chapter affect the operation of multiple utilities, functions,
 5526 and applications. There are other environment variables that are of interest only to specific
 5527 utilities. Environment variables that apply to a single utility only are defined as part of the
 5528 utility description. See the ENVIRONMENT VARIABLES section of the utility descriptions in
 5529 the Shell and Utilities volume of IEEE Std 1003.1-200x for information on environment variable
 5530 usage.

5531 The value of an environment variable is a string of characters. For a C-language program, an
 5532 array of strings called the environment shall be made available when a process begins. The array
 5533 is pointed to by the external variable *environ*, which is defined as:

```
5534     extern char **environ;
```

5535 These strings have the form *name=value*; *names* shall not contain the character '='. For values to
 5536 be portable across systems conforming to IEEE Std 1003.1-200x, the value shall be composed of
 5537 characters from the portable character set (except NUL and as indicated below). There is no
 5538 meaning associated with the order of strings in the environment. If more than one string in a
 5539 process' environment has the same *name*, the consequences are undefined.

5540 Environment variable names used by the utilities in the Shell and Utilities volume of
 5541 IEEE Std 1003.1-200x consist solely of uppercase letters, digits, and the '_' (underscore) from
 5542 the characters defined in Table 6-1 (on page 111) and do not begin with a digit. Other characters
 5543 may be permitted by an implementation; applications shall tolerate the presence of such names.
 5544 Uppercase and lowercase letters shall retain their unique identities and shall not be folded
 5545 together. The name space of environment variable names containing lowercase letters is
 5546 reserved for applications. Applications can define any environment variables with names from
 5547 this name space without modifying the behavior of the standard utilities.

5548 **Note:** Other applications may have difficulty dealing with environment variable names that start
 5549 with a digit. For this reason, use of such names is not recommended anywhere.

5550 The *values* that the environment variables may be assigned are not restricted except that they are
 5551 considered to end with a null byte and the total space used to store the environment and the
 5552 arguments to the process is limited to {ARG_MAX} bytes.

5553 Other *name=value* pairs may be placed in the environment by, for example, calling any of the
 5554 XSI *setenv()*, *unsetenv()*, or *putenv()* functions, manipulating the *environ* variable, or by using *envp*
 5555 arguments when creating a process; see *exec* in the System Interfaces volume of
 5556 IEEE Std 1003.1-200x.

5557 It is unwise to conflict with certain variables that are frequently exported by widely used
 5558 command interpreters and applications:

5559				
5560		ARFLAGS	IFS	MAILPATH
5561		CC	LANG	MAILRC
5562		CDPATH	LC_ALL	MAKEFLAGS
5563		CFLAGS	LC_COLLATE	MAKESHELL
5564		CHARSET	LC_CTYPE	MANPATH
5565		COLUMNS	LC_MESSAGES	MBOX
5566		DATMSK	LC_MONETARY	MORE
5567		DEAD	LC_NUMERIC	MSGVERB
5568		EDITOR	LC_TIME	NLSPATH
5569		ENV	LDFLAGS	NPROC
5570		EXINIT	LEX	OLDPWD
5571		FC	LFLAGS	OPTARG
5572		FCEDIT	LINENO	OPTERR
5573		FFLAGS	LINES	OPTIND
5574		GET	LISTER	PAGER
5575		GFLAGS	LOGNAME	PATH
5576		HISTFILE	LPDEST	PPID
5577		HISTORY	MAIL	PRINTER
5578		HISTSIZE	MAILCHECK	PROCLANG
5579		HOME	MAILER	PROJECTDIR

5580 If the variables in the following two sections are present in the environment during the
 5581 execution of an application or utility, they shall be given the meaning described below. Some are
 5582 placed into the environment by the implementation at the time the user logs in; all can be added
 5583 or changed by the user or any ancestor of the current process. The implementation adds or
 5584 changes environment variables named in IEEE Std 1003.1-200x only as specified in
 5585 IEEE Std 1003.1-200x. If they are defined in the application's environment, the utilities in the
 5586 Shell and Utilities volume of IEEE Std 1003.1-200x and the functions in the System Interfaces
 5587 volume of IEEE Std 1003.1-200x assume they have the specified meaning. Conforming
 5588 applications shall not set these environment variables to have meanings other than as described.
 5589 See *getenv()* and the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.12, Shell
 5590 Execution Environment for methods of accessing these variables.

5591 8.2 Internationalization Variables

5592 This section describes environment variables that are relevant to the operation of
 5593 internationalized interfaces described in IEEE Std 1003.1-200x.

5594 Users may use the following environment variables to announce specific localization
 5595 requirements to applications. Applications can retrieve this information using the *setlocale()*
 5596 function to initialize the correct behavior of the internationalized interfaces. The descriptions of
 5597 the internationalization environment variables describe the resulting behavior only when the
 5598 application locale is initialized in this way. The use of the internationalization variables by
 5599 utilities described in the Shell and Utilities volume of IEEE Std 1003.1-200x are described in the
 5600 ENVIRONMENT VARIABLES section for those utilities in addition to the global effects
 5601 described in this section.

5602 *LANG* This variable shall determine the locale category for native language, local
 5603 customs, and coded character set in the absence of the *LC_ALL* and other *LC_**
 5604 (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*,
 5605 *LC_TIME*) environment variables. This can be used by applications to
 5606 determine the language to use for error messages and instructions, collating
 5607 sequences, date formats, and so on.

5608	<i>LC_ALL</i>	This variable shall determine the values for all locale categories. The value of the <i>LC_ALL</i> environment variable has precedence over any of the other environment variables starting with <i>LC_(LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, LC_TIME)</i> and the <i>LANG</i> environment variable.
5609		
5610		
5611		
5612		
5613	<i>LC_COLLATE</i>	This variable shall determine the locale category for character collation. It determines collation information for regular expressions and sorting, including equivalence classes and multi-character collating elements, in various utilities and the <i>strcoll()</i> and <i>strxfrm()</i> functions. Additional semantics of this variable, if any, are implementation-defined.
5614		
5615		
5616		
5617		
5618	<i>LC_CTYPE</i>	This variable shall determine the locale category for character handling functions, such as <i>tolower()</i> , <i>toupper()</i> , and <i>isalpha()</i> . This environment variable determines the interpretation of sequences of bytes of text data as characters (for example, single as opposed to multi-byte characters), the classification of characters (for example, alpha, digit, graph), and the behavior of character classes. Additional semantics of this variable, if any, are implementation-defined.
5619		
5620		
5621		
5622		
5623		
5624		
5625	<i>LC_MESSAGES</i>	This variable shall determine the locale category for processing affirmative and negative responses and the language and cultural conventions in which messages should be written. It also affects the behavior of the <i>catopen()</i> function in determining the message catalog. Additional semantics of this variable, if any, are implementation-defined. The language and cultural conventions of diagnostic and informative messages whose format is unspecified by IEEE Std 1003.1-200x should be affected by the setting of <i>LC_MESSAGES</i> .
5626		
5627	XSI	
5628		
5629		
5630		
5631		
5632		
5633	<i>LC_MONETARY</i>	This variable shall determine the locale category for monetary-related numeric formatting information. Additional semantics of this variable, if any, are implementation-defined.
5634		
5635		
5636	<i>LC_NUMERIC</i>	This variable shall determine the locale category for numeric formatting (for example, thousands separator and radix character) information in various utilities as well as the formatted I/O operations in <i>printf()</i> and <i>scanf()</i> and the string conversion functions in <i>strtod()</i> . Additional semantics of this variable, if any, are implementation-defined.
5637		
5638		
5639		
5640		
5641	<i>LC_TIME</i>	This variable shall determine the locale category for date and time formatting information. It affects the behavior of the time functions in <i>strftime()</i> . Additional semantics of this variable, if any, are implementation-defined.
5642		
5643		
5644	XSI	
5645	<i>NLSPATH</i>	This variable shall contain a sequence of templates that the <i>catopen()</i> function uses when attempting to locate message catalogs. Each template consists of an optional prefix, one or more conversion specifications, a filename, and an optional suffix.
5646		
5647		
5648		For example:
5649		<code>NLSPATH="/system/nlslib/%N.cat"</code>
5650		defines that <i>catopen()</i> should look for all message catalogs in the directory <code>/system/nlslib</code> , where the catalog name should be constructed from the <i>name</i> parameter passed to <i>catopen()</i> (<code>%N</code>), with the suffix <code>.cat</code> .
5651		
5652		
5653		Conversion specifications consist of a <code>'%'</code> symbol, followed by a single-letter keyword. The following keywords are currently defined:
5654		

5655	%N	The value of the <i>name</i> parameter passed to <i>catopen()</i> .
5656	%L	The value of the <i>LC_MESSAGES</i> category.
5657	%l	The <i>language</i> element from the <i>LC_MESSAGES</i> category.
5658	%t	The <i>territory</i> element from the <i>LC_MESSAGES</i> category.
5659	%c	The <i>codeset</i> element from the <i>LC_MESSAGES</i> category.
5660	%%	A single '%' character.
5661		An empty string is substituted if the specified value is not currently defined.
5662		The separators underscore ('_') and period ('.') are not included in the %t
5663		and %c conversion specifications.
5664		Templates defined in <i>NLSPATH</i> are separated by colons (':'). A leading or
5665		two adjacent colons "::" is equivalent to specifying %N. For example:
5666		<code>NLSPATH=":%N.cat:/nlslib/%L/%N.cat"</code>
5667		indicates to <i>catopen()</i> that it should look for the requested message catalog in
5668		<i>name</i> , <i>name.cat</i> , and <i>/nlslib/category/name.cat</i> , where <i>category</i> is the value of the
5669		<i>LC_MESSAGES</i> category of the current locale.
5670		Users should not set the <i>NLSPATH</i> variable unless they have a specific reason
5671		to override the default system path. Setting <i>NLSPATH</i> to override the default
5672		system path produces undefined results in the standard utilities and in
5673		applications with appropriate privileges.
5674		The environment variables <i>LANG</i> , <i>LC_ALL</i> , <i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> ,
5675	XSI	<i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i> , and <i>NLSPATH</i> provide for the support of
5676		internationalized applications. The standard utilities shall make use of these environment
5677		variables as described in this section and the individual ENVIRONMENT VARIABLES sections
5678		for the utilities. If these variables specify locale categories that are not based upon the same
5679		underlying codeset, the results are unspecified.
5680		The values of locale categories shall be determined by a precedence order; the first condition met
5681		below determines the value:
5682		1. If the <i>LC_ALL</i> environment variable is defined and is not null, the value of <i>LC_ALL</i> shall be
5683		used.
5684		2. If the <i>LC_*</i> environment variable (<i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> ,
5685		<i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i>) is defined and is not null, the value of the
5686		environment variable shall be used to initialize the category that corresponds to the
5687		environment variable.
5688		3. If the <i>LANG</i> environment variable is defined and is not null, the value of the <i>LANG</i>
5689		environment variable shall be used.
5690		4. If the <i>LANG</i> environment variable is not set or is set to the empty string, the
5691		implementation-defined default locale shall be used.
5692		If the locale value is "C" or "POSIX", the POSIX locale shall be used and the standard utilities
5693		behave in accordance with the rules in Section 7.2 (on page 120) for the associated category.
5694		If the locale value begins with a slash, it shall be interpreted as the pathname of a file that was
5695		created in the output format used by the <i>localedef</i> utility; see OUTPUT FILES under <i>localedef</i> .
5696		Referencing such a pathname shall result in that locale being used for the indicated category.

5697 XSI If the locale value has the form:

5698 `language[_territory][.codeset]`

5699 it refers to an implementation-provided locale, where settings of language, territory, and codeset
5700 are implementation-defined.

5701 `LC_COLLATE`, `LC_CTYPE`, `LC_MESSAGES`, `LC_MONETARY`, `LC_NUMERIC`, and `LC_TIME` are
5702 defined to accept an additional field `@modifier`, which allows the user to select a specific instance
5703 of localization data within a single category (for example, for selecting the dictionary as opposed
5704 to the character ordering of data). The syntax for these environment variables is thus defined as:

5705 `[language[_territory][.codeset][@modifier]]`

5706 For example, if a user wanted to interact with the system in French, but required to sort German
5707 text files, `LANG` and `LC_COLLATE` could be defined as:

5708 `LANG=Fr_FR`
5709 `LC_COLLATE=De_DE`

5710 This could be extended to select dictionary collation (say) by use of the `@modifier` field; for
5711 example:

5712 `LC_COLLATE=De_DE@dict`

5713

5714 An implementation may support other formats.

5715 If the locale value is not recognized by the implementation, the behavior is unspecified.

5716 At runtime, these values are bound to a program's locale by calling the `setlocale()` function.

5717 Additional criteria for determining a valid locale name are implementation-defined.

5718 8.3 Other Environment Variables

5719 **`COLUMNS`** This variable shall represent a decimal integer >0 used to indicate the user's
5720 preferred width in column positions for the terminal screen or window; see
5721 Section 3.103 (on page 47). If this variable is unset or null, the implementation
5722 determines the number of columns, appropriate for the terminal or window,
5723 in an unspecified manner. When `COLUMNS` is set, any terminal-width
5724 information implied by `TERM` is overridden. Users and conforming |
5725 applications should not set `COLUMNS` unless they wish to override the |
5726 system selection and produce output unrelated to the terminal characteristics.

5727 Users should not need to set this variable in the environment unless there is a
5728 specific reason to override the implementation's default behavior, such as to
5729 display data in an area arbitrarily smaller than the terminal or window.

5730 XSI **`DATMSK`** Indicates the pathname of the template file used by `getdate()`.

5731 **`HOME`** The system shall initialize this variable at the time of login to be a pathname of
5732 the user's home directory. See <`pwd.h`>.

5733 **`LINES`** This variable shall represent a decimal integer >0 used to indicate the user's
5734 preferred number of lines on a page or the vertical screen or window size in
5735 lines. A line in this case is a vertical measure large enough to hold the tallest
5736 character in the character set being displayed. If this variable is unset or null,
5737 the implementation determines the number of lines, appropriate for the

5738		terminal or window (size, terminal baud rate, and so on), in an unspecified manner. When <i>LINES</i> is set, any terminal-height information implied by <i>TERM</i> is overridden. Users and conforming applications should not set <i>LINES</i> unless they wish to override the system selection and produce output unrelated to the terminal characteristics.
5739		
5740		
5741		
5742		
5743		Users should not need to set this variable in the environment unless there is a specific reason to override the implementation's default behavior, such as to display data in an area arbitrarily smaller than the terminal or window.
5744		
5745		
5746	<i>LOGNAME</i>	The system shall initialize this variable at the time of login to be the user's login name. See < <i>pwd.h</i> >. For a value of <i>LOGNAME</i> to be portable across implementations of IEEE Std 1003.1-200x, the value should be composed of characters from the portable filename character set.
5747		
5748		
5749		
5750	XSI <i>MSGVERB</i>	Describes which message components shall be used in writing messages by <i>fmtmsg()</i> .
5751		
5752	<i>PATH</i>	This variable shall represent the sequence of path prefixes that certain functions and utilities apply in searching for an executable file known only by a filename. The prefixes shall be separated by a colon (' : '). When a non-zero-length prefix is applied to this filename, a slash shall be inserted between the prefix and the filename. A zero-length prefix is a legacy feature that indicates the current working directory. It appears as two adjacent colons (" : : "), as an initial colon preceding the rest of the list, or as a trailing colon following the rest of the list. A strictly conforming application shall use an actual pathname (such as <i>.</i>) to represent the current working directory in <i>PATH</i> . The list shall be searched from beginning to end, applying the filename to each prefix, until an executable file with the specified name and appropriate execution permissions is found. If the pathname being sought contains a slash, the search through the path prefixes shall not be performed. If the pathname begins with a slash, the specified path is resolved (see Section 4.11 (on page 98)). If <i>PATH</i> is unset or is set to null, the path search is implementation-defined.
5753		
5754		
5755		
5756		
5757		
5758		
5759		
5760		
5761		
5762		
5763		
5764		
5765		
5766		
5767		
5768	<i>PWD</i>	This variable shall represent an absolute pathname of the current working directory. It shall not contain any filename components of dot or dot-dot. The value is set by the <i>cd</i> utility.
5769		
5770		
5771	<i>SHELL</i>	This variable shall represent a pathname of the user's preferred command language interpreter. If this interpreter does not conform to the Shell Command Language in the Shell and Utilities volume of IEEE Std 1003.1-200x, Chapter 2, Shell Command Language, utilities may behave differently from those described in IEEE Std 1003.1-200x.
5772		
5773		
5774		
5775		
5776	<i>TMPDIR</i>	This variable shall represent a pathname of a directory made available for programs that need a place to create temporary files.
5777		
5778	<i>TERM</i>	This variable shall represent the terminal type for which output is to be prepared. This information is used by utilities and application programs wishing to exploit special capabilities specific to a terminal. The format and allowable values of this environment variable are unspecified.
5779		
5780		
5781		
5782	<i>TZ</i>	This variable shall represent timezone information. The contents of the environment variable named <i>TZ</i> shall be used by the <i>ctime()</i> , <i>localtime()</i> , <i>strftime()</i> , and <i>mktime()</i> functions, and by various utilities, to override the default timezone. The value of <i>TZ</i> has one of the two forms (spaces inserted
5783		
5784		
5785		

5786 for clarity):

5787 : *characters*

5788 or:

5789 *std offset dst offset, rule*

5790 If *TZ* is of the first format (that is, if the first character is a colon), the
5791 characters following the colon are handled in an implementation-defined
5792 manner.

5793 The expanded format (for all *TZs* whose value does not have a colon as the
5794 first character) is as follows:

5795 *stdoffset[dst[offset][,start[/time],end[/time]]]*

5796 Where:

5797 *std* and *dst* Indicate no less than three, nor more than {TZNAME_MAX},
5798 bytes that are the designation for the standard (*std*) or the
5799 alternative (*dst*—such as Daylight Savings Time) timezone. Only
5800 *std* is required; if *dst* is missing, then the alternative time does
5801 not apply in this locale.

5802 Each of these fields may occur in either of two formats quoted or
5803 unquoted:

5804 — In the quoted form, the first character shall be the less-than
5805 ('<') character and the last character shall be the greater-than
5806 ('>') character. All characters between these quoting
5807 characters shall be alphanumeric characters in the current
5808 locale, the plus-sign ('+') character, or the minus-sign ('-')
5809 character. The *std* and *dst* fields in this case shall not include
5810 the quoting characters. |

5811 — In the unquoted form, all characters in these fields shall be
5812 alphabetic characters in the current locale.

5813 The interpretation of these fields is unspecified if either field is
5814 less than three bytes (except for the case when *dst* is missing),
5815 more than {TZNAME_MAX} bytes, or if they contain characters
5816 other than those specified.

5817 *offset* Indicates the value added to the local time to arrive at
5818 Coordinated Universal Time. The *offset* has the form:

5819 *hh[:mm[:ss]]*

5820 The minutes (*mm*) and seconds (*ss*) are optional. The hour (*hh*)
5821 shall be required and may be a single digit. The *offset* following
5822 *std* shall be required. If no *offset* follows *dst*, the alternative time
5823 is assumed to be one hour ahead of standard time. One or more
5824 digits may be used; the value is always interpreted as a decimal
5825 number. The hour shall be between zero and 24, and the minutes
5826 (and seconds)—if present—between zero and 59. The result of
5827 using values outside of this range is unspecified. If preceded by
5828 a '-', the timezone shall be east of the Prime Meridian;
5829 otherwise, it shall be west (which may be indicated by an
5830 optional preceding '+').

5831 *rule* Indicates when to change to and back from the alternative time.
5832 The *rule* has the form:
5833 $date[/time], date[/time]$
5834 where the first *date* describes when the change from standard to
5835 alternative time occurs and the second *date* describes when the
5836 change back happens. Each *time* field describes when, in current
5837 local time, the change to the other time is made.
5838 The format of *date* is one of the following:
5839 *Jn* The Julian day n ($1 \leq n \leq 365$). Leap days shall not be
5840 counted. That is, in all years—including leap years—
5841 February 28 is day 59 and March 1 is day 60. It is
5842 impossible to refer explicitly to the occasional February
5843 29.
5844 *n* The zero-based Julian day ($0 \leq n \leq 365$). Leap days shall
5845 be counted, and it is possible to refer to February 29.
5846 *Mm.n.d* The d 'th day ($0 \leq d \leq 6$) of week n of month m of the
5847 year ($1 \leq n \leq 5$, $1 \leq m \leq 12$, where week 5 means “the
5848 last d day in month m ” which may occur in either the
5849 fourth or the fifth week). Week 1 is the first week in
5850 which the d 'th day occurs. Day zero is Sunday.
5851 The *time* has the same format as *offset* except that no leading sign
5852 ('-' or '+') is allowed. The default, if *time* is not given, shall be
5853 02:00:00.

Regular Expressions

5854

5855 *Regular Expressions* (REs) provide a mechanism to select specific strings from a set of character
5856 strings.

5857 Regular expressions are a context-independent syntax that can represent a wide variety of
5858 character sets and character set orderings, where these character sets are interpreted according
5859 to the current locale. While many regular expressions can be interpreted differently depending
5860 on the current locale, many features, such as character class expressions, provide for contextual
5861 invariance across locales.

5862 The Basic Regular Expression (BRE) notation and construction rules in Section 9.3 (on page 167)
5863 shall apply to most utilities supporting regular expressions. Some utilities, instead, support the
5864 Extended Regular Expressions (ERE) described in Section 9.4 (on page 171); any exceptions for
5865 both cases are noted in the descriptions of the specific utilities using regular expressions. Both
5866 BREs and ERAs are supported by the Regular Expression Matching interface in the System
5867 Interfaces volume of IEEE Std 1003.1-200x under *regcomp()*, *regex()*, and related functions.

5868 9.1 Regular Expression Definitions

5869 For the purposes of this section, the following definitions shall apply:

5870 **entire regular expression**

5871 The concatenated set of one or more BREs or ERAs that make up the pattern specified for
5872 string selection.

5873 **matched**

5874 A sequence of zero or more characters shall be said to be matched by a BRE or ERA when
5875 the characters in the sequence correspond to a sequence of characters defined by the
5876 pattern.

5877 Matching shall be based on the bit pattern used for encoding the character, not on the
5878 graphic representation of the character. This means that if a character set contains two or
5879 more encodings for a graphic symbol, or if the strings searched contain text encoded in
5880 more than one codeset, no attempt is made to search for any other representation of the
5881 encoded symbol. If that is required, the user can specify equivalence classes containing all
5882 variations of the desired graphic symbol.

5883 The search for a matching sequence starts at the beginning of a string and stops when the
5884 first sequence matching the expression is found, where *first* is defined to mean “begins
5885 earliest in the string”. If the pattern permits a variable number of matching characters and
5886 thus there is more than one such sequence starting at that point, the longest such sequence
5887 is matched. For example: the BRE "bb*" matches the second to fourth characters of *abbbc*,
5888 and the ERA (*wee* | *week*)(*knights* | *night*) matches all ten characters of *weeknights*.

5889 Consistent with the whole match being the longest of the leftmost matches, each subpattern,
5890 from left to right, shall match the longest possible string. For this purpose, a null string shall
5891 be considered to be longer than no match at all. For example, matching the BRE
5892 "\(.*\).*" against "abcdef", the subexpression "(\1)" is "abcdef", and matching
5893 the BRE "\(a*\).*" against "bc", the subexpression "(\1)" is the null string.

5894 When a multi-character collating element in a bracket expression (see Section 9.3.5 (on page
5895 168)) is involved, the longest sequence shall be measured in characters consumed from the

5896 string to be matched; that is, the collating element counts not as one element, but as the
5897 number of characters it matches.

5898 **BRE (ERE) matching a single character**

5899 A BRE or ERE that shall match either a single character or a single collating element.

5900 Only a BRE or ERE of this type that includes a bracket expression (see Section 9.3.5 (on page
5901 168)) can match a collating element.

5902 **BRE (ERE) matching multiple characters**

5903 A BRE or ERE that shall match a concatenation of single characters or collating elements.

5904 Such a BRE or ERE is made up from a BRE (ERE) matching a single character and BRE (ERE)
5905 special characters.

5906 **invalid**

5907 This section uses the term *invalid* for certain constructs or conditions. Invalid REs shall
5908 cause the utility or function using the RE to generate an error condition. When *invalid* is not
5909 used, violations of the specified syntax or semantics for REs produce undefined results: this
5910 may entail an error, enabling an extended syntax for that RE, or using the construct in error
5911 as literal characters to be matched. For example, the BRE construct "`\{1,2,3\}`" does not
5912 comply with the grammar. A conforming application cannot rely on it producing an error
5913 nor matching the literal characters "`\{1,2,3\}`".

5914 **9.2 Regular Expression General Requirements**

5915 The requirements in this section shall apply to both basic and extended regular expressions.

5916 The use of regular expressions is generally associated with text processing. REs (BREs and EREs)
5917 operate on text strings; that is, zero or more characters followed by an end-of-string delimiter
5918 (typically NUL). Some utilities employing regular expressions limit the processing to lines; that
5919 is, zero or more characters followed by a <newline>. In the regular expression processing
5920 described in IEEE Std 1003.1-200x, the <newline> is regarded as an ordinary character and both a
5921 period and a non-matching list can match one. The Shell and Utilities volume of
5922 IEEE Std 1003.1-200x specifies within the individual descriptions of those standard utilities
5923 employing regular expressions whether they permit matching of <newline>s; if not stated
5924 otherwise, the use of literal <newline>s or any escape sequence equivalent produces undefined
5925 results. Those utilities (like *grep*) that do not allow <newline>s to match are responsible for
5926 eliminating any <newline> from strings before matching against the RE. The *regcomp()* function
5927 in the System Interfaces volume of IEEE Std 1003.1-200x, however, can provide support for such
5928 processing without violating the rules of this section.

5929 The interfaces specified in IEEE Std 1003.1-200x do not permit the inclusion of a NUL character
5930 in an RE or in the string to be matched. If during the operation of a standard utility a NUL is
5931 included in the text designated to be matched, that NUL may designate the end of the text string
5932 for the purposes of matching.

5933 When a standard utility or function that uses regular expressions specifies that pattern matching
5934 shall be performed without regard to the case (uppercase or lowercase) of either data or
5935 patterns, then when each character in the string is matched against the pattern, not only the
5936 character, but also its case counterpart (if any), shall be matched. This definition of case-
5937 insensitive processing is intended to allow matching of multi-character collating elements as
5938 well as characters, as each character in the string is matched using both its cases. For example,
5939 in a locale where "Ch" is a multi-character collating element and where a matching list expression
5940 matches such elements, the RE "[[.Ch.]]" when matched against the string "char", is in

5941 reality matched against "ch", "Ch", "cH", and "CH".
 5942 The implementation shall support any regular expression that does not exceed 256 bytes in
 5943 length.

5944 **9.3 Basic Regular Expressions**

5945 **9.3.1 BREs Matching a Single Character or Collating Element**

5946 A BRE ordinary character, a special character preceded by a backslash or a period, shall match a
 5947 single character. A bracket expression shall match a single character or a single collating
 5948 element.

5949 **9.3.2 BRE Ordinary Characters**

5950 An ordinary character is a BRE that matches itself: any character in the supported character set,
 5951 except for the BRE special characters listed in Section 9.3.3.

5952 The interpretation of an ordinary character preceded by a backslash ('**') is undefined, except
 5953 for:

- 5954 • The characters ') ', ' (', ' { ', and ' } '
- 5955 • The digits 1 to 9 inclusive (see Section 9.3.6 (on page 170))
- 5956 • A character inside a bracket expression

5957 **9.3.3 BRE Special Characters**

5958 A *BRE special character* has special properties in certain contexts. Outside those contexts, or when
 5959 preceded by a backslash, such a character is a BRE that matches the special character itself. The
 5960 BRE special characters and the contexts in which they have their special meaning are as follows:

5961 . [** The period, left-bracket, and backslash shall be special except when used in a bracket
 5962 expression (see Section 9.3.5 (on page 168)). An expression containing a '[' that is not
 5963 preceded by a backslash and is not part of a bracket expression produces undefined
 5964 results.

5965 * The asterisk shall be special except when used:

- 5966 • In a bracket expression
- 5967 • As the first character of an entire BRE (after an initial '^', if any)
- 5968 • As the first character of a subexpression (after an initial '^', if any); see Section
 5969 9.3.6 (on page 170)

5970 ^ The circumflex shall be special when used as:

- 5971 • An anchor (see Section 9.3.8 (on page 171))
- 5972 • The first character of a bracket expression (see Section 9.3.5 (on page 168))

5973 \$ The dollar sign shall be special when used as an anchor.

5974 **9.3.4 Periods in BREs**

5975 A period (' . '), when used outside a bracket expression, is a BRE that shall match any character
5976 in the supported character set except NUL.

5977 **9.3.5 RE Bracket Expression**

5978 A bracket expression (an expression enclosed in square brackets, " [] ") is an RE that shall match |
5979 a single collating element contained in the non-empty set of collating elements represented by |
5980 the bracket expression.

5981 The following rules and definitions apply to bracket expressions:

5982 1. A *bracket expression* is either a matching list expression or a non-matching list expression. It
5983 consists of one or more expressions: collating elements, collating symbols, equivalence
5984 classes, character classes, or range expressions. The right-bracket ('] ') shall lose its special
5985 meaning and represents itself in a bracket expression if it occurs first in the list (after an
5986 initial circumflex (' ^ '), if any). Otherwise, it shall terminate the bracket expression, unless
5987 it appears in a collating symbol (such as " [.] .] ") or is the ending right-bracket for a
5988 collating symbol, equivalence class, or character class. The special characters ' . ' , ' * ' ,
5989 ' [' , and ' \ ' (period, asterisk, left-bracket, and backslash, respectively) shall lose their
5990 special meaning within a bracket expression.

5991 The character sequences " [. " , " [= " , and " [: " (left-bracket followed by a period, equals-
5992 sign, or colon) shall be special inside a bracket expression and are used to delimit collating
5993 symbols, equivalence class expressions, and character class expressions. These symbols
5994 shall be followed by a valid expression and the matching terminating sequence " .] " ,
5995 "=] " , or " :] " , as described in the following items.

5996 2. A *matching list expression* specifies a list that shall match any single-character collating |
5997 element in any of the expressions represented in the list. The first character in the list shall |
5998 not be the circumflex; for example, "[abc] " is an RE that matches any of the characters |
5999 ' a ' , ' b ' , or ' c ' . It is unspecified whether a matching list expression matches a multi-
6000 character collating element that is matched by one of the expressions.

6001 3. A *non-matching list expression* begins with a circumflex (' ^ '), and specifies a list that shall |
6002 match any single-character collating element except for the expressions represented in the |
6003 list after the leading circumflex. For example, "[^ abc] " is an RE that matches any |
6004 character except the characters ' a ' , ' b ' , or ' c ' . It is unspecified whether a non-matching
6005 list expression matches a multi-character collating element that is not matched by any of
6006 the expressions. The circumflex shall have this special meaning only when it occurs first in
6007 the list, immediately following the left-bracket.

6008 4. A *collating symbol* is a collating element enclosed within bracket-period (" [. " and " .] ") |
6009 delimiters. Collating elements are defined as described in Section 7.3.2.4 (on page 133). |
6010 Conforming applications shall represent multi-character collating elements as collating |
6011 symbols when it is necessary to distinguish them from a list of the individual characters |
6012 that make up the multi-character collating element. For example, if the string " ch " is a
6013 collating element defined using the line:

```
6014 collating-element <ch-digraph> from "<c><h>"
```

6015 in the locale definition, the expression "[[. ch.]] " shall be treated as an RE containing
6016 the collating symbol ' ch ' , while "[ch] " shall be treated as an RE matching ' c ' or ' h ' .
6017 Collating symbols are recognized only inside bracket expressions. If the string is not a
6018 collating element in the current locale, the expression is invalid.

6019 5. An *equivalence class expression* shall represent the set of collating elements belonging to an
 6020 equivalence class, as described in Section 7.3.2.4 (on page 133). Only primary equivalence
 6021 classes shall be recognized. The class shall be expressed by enclosing any one of the
 6022 collating elements in the equivalence class within bracket-equal ("[" and "=")
 6023 delimiters. For example, if 'a', 'à', and 'â' belong to the same equivalence class, then
 6024 "[[=a=]b]", "[[=à=]b]", and "[[=â=]b]" are each equivalent to "[aââb]". If the
 6025 collating element does not belong to an equivalence class, the equivalence class expression
 6026 shall be treated as a *collating symbol*.

6027 6. A *character class expression* shall represent the union of two sets: |
 6028 a. The set of single-character collating elements whose characters belong to the |
 6029 character class, as defined in the *LC_CTYPE* category in the current locale. |
 6030 b. An unspecified set of multi-character collating elements. |

6031 All character classes specified in the current locale shall be recognized. A character class |
 6032 expression is expressed as a character class name enclosed within bracket-colon ("[" and |
 6033 ":") delimiters.

6034 The following character class expressions shall be supported in all locales:

```
6035      [:alnum:]      [:cntrl:]      [:lower:]      [:space:]
6036      [:alpha:]      [:digit:]      [:print:]      [:upper:]
6037      [:blank:]      [:graph:]      [:punct:]      [:xdigit:]
```

6038 XSI In addition, character class expressions of the form:

```
6039      [:name:]
```

6040 are recognized in those locales where the *name* keyword has been given a **charclass**
 6041 definition in the *LC_CTYPE* category.

6042 7. In the POSIX locale, a range expression represents the set of collating elements that fall
 6043 between two elements in the collation sequence, inclusive. In other locales, a range
 6044 expression has unspecified behavior: strictly conforming applications shall not rely on
 6045 whether the range expression is valid, or on the set of collating elements matched. A range
 6046 expression shall be expressed as the starting point and the ending point separated by a
 6047 hyphen ('-').

6048 In the following, all examples assume the POSIX locale.

6049 The starting range point and the ending range point shall be a collating element or
 6050 collating symbol. An equivalence class expression used as a starting or ending point of a
 6051 range expression produces unspecified results. An equivalence class can be used portably
 6052 within a bracket expression, but only outside the range. If the represented set of collating
 6053 elements is empty, it is unspecified whether the expression matches nothing, or is treated
 6054 as invalid.

6055 The interpretation of range expressions where the ending range point is also the starting
 6056 range point of a subsequent range expression (for example, "[a-m-o]") is undefined.

6057 The hyphen character shall be treated as itself if it occurs first (after an initial '^', if any)
 6058 or last in the list, or as an ending range point in a range expression. As examples, the
 6059 expressions "[ac]" and "[ac-]" are equivalent and match any of the characters 'a',
 6060 'c', or '-'; "[^ac]" and "[^ac-]" are equivalent and match any characters except
 6061 'a', 'c', or '-'; the expression "[%--]" matches any of the characters between '%' and
 6062 '-' inclusive; the expression "[--@]" matches any of the characters between '-' and
 6063 '@' inclusive; and the expression "[a--@]" is either invalid or equivalent to '@',

6064 because the letter 'a' follows the symbol '-' in the POSIX locale. To use a hyphen as the
 6065 starting range point, it shall either come first in the bracket expression or be specified as a
 6066 collating symbol; for example, "[] [. -] - 0]", which matches either a right bracket or
 6067 any character or collating element that collates between hyphen and 0, inclusive.

6068 If a bracket expression specifies both '-' and ']', the ']' shall be placed first (after the
 6069 '^', if any) and the '-' last within the bracket expression.

6070 9.3.6 BREs Matching Multiple Characters

6071 The following rules can be used to construct BREs matching multiple characters from BREs
 6072 matching a single character:

6073 1. The concatenation of BREs shall match the concatenation of the strings matched by each
 6074 component of the BRE.

6075 2. A *subexpression* can be defined within a BRE by enclosing it between the character pairs
 6076 "\(" and "\)". Such a subexpression shall match whatever it would have matched
 6077 without the "\(" and "\)", except that anchoring within subexpressions is optional
 6078 behavior; see Section 9.3.8 (on page 171). Subexpressions can be arbitrarily nested.

6079 3. The *back-reference expression* '\n' shall match the same (possibly empty) string of |
 6080 characters as was matched by a subexpression enclosed between "\(" and "\)"
 6081 preceding the '\n'. The character 'n' shall be a digit from 1 through 9, specifying the
 6082 *n*th subexpression (the one that begins with the *n*th "\(" from the beginning of the
 6083 pattern and ends with the corresponding paired "\)"). The expression is invalid if less
 6084 than *n* subexpressions precede the '\n'. For example, the expression "\(.*)\1\$" |
 6085 matches a line consisting of two adjacent appearances of the same string, and the
 6086 expression "\(a\)*\1" fails to match 'a'. When the referenced subexpression matched
 6087 more than one string, the back-referenced expression shall refer to the last matched string.
 6088 If the subexpression referenced by the back-reference matches more than one string
 6089 because of an asterisk ('*') or an interval expression (see item (5)), the back-reference
 6090 shall match the last (rightmost) of these strings.

6091 4. When a BRE matching a single character, a subexpression, or a back-reference is followed
 6092 by the special character asterisk ('*'), together with that asterisk it shall match what zero
 6093 or more consecutive occurrences of the BRE would match. For example, "[ab]*" and
 6094 "[ab][ab]" are equivalent when matching the string "ab".

6095 5. When a BRE matching a single character, a subexpression, or a back-reference is followed
 6096 by an *interval expression* of the format "\{m\}", "\{m,\}", or "\{m,n\}", together with
 6097 that interval expression it shall match what repeated consecutive occurrences of the BRE
 6098 would match. The values of *m* and *n* are decimal integers in the range 0
 6099 $\leq m \leq n \leq \{RE_DUP_MAX\}$, where *m* specifies the exact or minimum number of occurrences
 6100 and *n* specifies the maximum number of occurrences. The expression "\{m\}" shall match
 6101 exactly *m* occurrences of the preceding BRE, "\{m,\}" shall match at least *m* occurrences,
 6102 and "\{m,n\}" shall match any number of occurrences between *m* and *n*, inclusive.

6103 For example, in the string "abababcccccd" the BRE "c\{3\}" is matched by
 6104 characters '7' to '9', the BRE "\(ab\)\{4,\}" is not matched at all, and the BRE
 6105 "c\{1,3\}d" is matched by characters ten to thirteen.

6106 The behavior of multiple adjacent duplication symbols ('*' and intervals) produces undefined
 6107 results.

6108 A subexpression repeated by an asterisk ('*') or an interval expression shall not match a null
 6109 expression unless this is the only match for the repetition or it is necessary to satisfy the exact or

6110 minimum number of occurrences for the interval expression.

6111 9.3.7 BRE Precedence

6112 The order of precedence shall be as shown in the following table:

BRE Precedence (from high to low)	
6113 Collation-related bracket symbols	[==] [::] [..]
6114 Escaped characters	\<special character>
6115 Bracket expression	[]
6116 Subexpressions/back-references	\(\) \n
6117 Single-character-BRE duplication	* \{m,n\}
6118 Concatenation	
6119 Anchoring	^ \$

6121 9.3.8 BRE Expression Anchoring

6122 A BRE can be limited to matching strings that begin or end a line; this is called *anchoring*. The
 6123 circumflex and dollar sign special characters shall be considered BRE anchors in the following
 6124 contexts:

- 6125 1. A circumflex ('^') shall be an anchor when used as the first character of an entire BRE.
 6126 The implementation may treat the circumflex as an anchor when used as the first character
 6127 of a subexpression. The circumflex shall anchor the expression (or optionally
 6128 subexpression) to the beginning of a string; only sequences starting at the first character of
 6129 a string shall be matched by the BRE. For example, the BRE "^ab" matches "ab" in the
 6130 string "abcdef", but fails to match in the string "cdefab". The BRE "\(^ab\)" may
 6131 match the former string. A portable BRE shall escape a leading circumflex in a
 6132 subexpression to match a literal circumflex.
- 6133 2. A dollar sign ('\$') shall be an anchor when used as the last character of an entire BRE.
 6134 The implementation may treat a dollar sign as an anchor when used as the last character of
 6135 a subexpression. The dollar sign shall anchor the expression (or optionally subexpression)
 6136 to the end of the string being matched; the dollar sign can be said to match the end-of-
 6137 string following the last character.
- 6138 3. A BRE anchored by both '^' and '\$' shall match only an entire string. For example, the
 6139 BRE "^abcdef\$" matches strings consisting only of "abcdef".

6140 9.4 Extended Regular Expressions

6141 The *extended regular expression* (ERE) notation and construction rules shall apply to utilities
 6142 defined as using extended regular expressions; any exceptions to the following rules are noted in
 6143 the descriptions of the specific utilities using EREs.

6144 **9.4.1 EREs Matching a Single Character or Collating Element**

6145 An ERE ordinary character, a special character preceded by a backslash, or a period shall match
 6146 a single character. A bracket expression shall match a single character or a single collating
 6147 element. An *ERE matching a single character* enclosed in parentheses shall match the same as the
 6148 ERE without parentheses would have matched.

6149 **9.4.2 ERE Ordinary Characters**

6150 An *ordinary character* is an ERE that matches itself. An ordinary character is any character in the
 6151 supported character set, except for the ERE special characters listed in Section 9.4.3. The
 6152 interpretation of an ordinary character preceded by a backslash ('**') is undefined.

6153 **9.4.3 ERE Special Characters**

6154 An *ERE special character* has special properties in certain contexts. Outside those contexts, or
 6155 when preceded by a backslash, such a character shall be an ERE that matches the special
 6156 character itself. The extended regular expression special characters and the contexts in which
 6157 they shall have their special meaning are as follows:

6158 . [** (The period, left-bracket, backslash, and left-parenthesis shall be special except when
 6159 used in a bracket expression (see Section 9.3.5 (on page 168)). Outside a bracket
 6160 expression, a left-parenthesis immediately followed by a right-parenthesis produces
 6161 undefined results.

6162) The right-parenthesis shall be special when matched with a preceding left-parenthesis,
 6163 both outside a bracket expression.

6164 * + ? { The asterisk, plus-sign, question-mark, and left-brace shall be special except when used
 6165 in a bracket expression (see Section 9.3.5 (on page 168)). Any of the following uses
 6166 produce undefined results:

- 6167 • If these characters appear first in an ERE, or immediately following a vertical-line,
 6168 circumflex, or left-parenthesis

- 6169 • If a left-brace is not part of a valid interval expression (see Section 9.4.6 (on page
 6170 173))

6171 | The vertical-line is special except when used in a bracket expression (see Section 9.3.5
 6172 (on page 168)). A vertical-line appearing first or last in an ERE, or immediately
 6173 following a vertical-line or a left-parenthesis, or immediately preceding a right-
 6174 parenthesis, produces undefined results.

6175 ^ The circumflex shall be special when used as:

- 6176 • An anchor (see Section 9.4.9 (on page 174))

- 6177 • The first character of a bracket expression (see Section 9.3.5 (on page 168))

6178 \$ The dollar sign shall be special when used as an anchor.

6179 **9.4.4 Periods in EREs**

6180 A period (`'.'`), when used outside a bracket expression, is an ERE that shall match any
6181 character in the supported character set except NUL.

6182 **9.4.5 ERE Bracket Expression**

6183 The rules for ERE Bracket Expressions are the same as for Basic Regular Expressions; see Section
6184 9.3.5 (on page 168).

6185 **9.4.6 EREs Matching Multiple Characters**

6186 The following rules shall be used to construct EREs matching multiple characters from EREs
6187 matching a single character:

- 6188 1. A *concatenation of EREs* shall match the concatenation of the character sequences matched
6189 by each component of the ERE. A concatenation of EREs enclosed in parentheses shall
6190 match whatever the concatenation without the parentheses matches. For example, both the
6191 ERE `"cd"` and the ERE `"(cd)"` are matched by the third and fourth character of the string
6192 `"abcdefabcdef"`.
- 6193 2. When an ERE matching a single character or an ERE enclosed in parentheses is followed by
6194 the special character plus-sign (`'+'`), together with that plus-sign it shall match what one
6195 or more consecutive occurrences of the ERE would match. For example, the ERE
6196 `"b+(bc)"` matches the fourth to seventh characters in the string `"acabbbbcde"`. And,
6197 `"[ab]+"` and `"[ab][ab]*"` are equivalent.
- 6198 3. When an ERE matching a single character or an ERE enclosed in parentheses is followed by
6199 the special character asterisk (`'*'`), together with that asterisk it shall match what zero or
6200 more consecutive occurrences of the ERE would match. For example, the ERE `"b*c"`
6201 matches the first character in the string `"cabbbbcde"`, and the ERE `"b*cd"` matches the
6202 third to seventh characters in the string `"cabbbbcdebbbbbbbcdbc"`. And, `"[ab]*"` and
6203 `[ab][ab]` are equivalent when matching the string `"ab"`.
- 6204 4. When an ERE matching a single character or an ERE enclosed in parentheses is followed by
6205 the special character question-mark (`'?'`), together with that question-mark it shall match
6206 what zero or one consecutive occurrences of the ERE would match. For example, the ERE
6207 `"b?c"` matches the second character in the string `"acabbbbcde"`.
- 6208 5. When an ERE matching a single character or an ERE enclosed in parentheses is followed by
6209 an *interval expression* of the format `"{m}"`, `"{m,}"`, or `"{m,n}"`, together with that
6210 interval expression it shall match what repeated consecutive occurrences of the ERE would
6211 match. The values of *m* and *n* are decimal integers in the range $0 \leq m \leq n \leq \{RE_DUP_MAX\}$,
6212 where *m* specifies the exact or minimum number of occurrences and *n* specifies the
6213 maximum number of occurrences. The expression `"{m}"` matches exactly *m* occurrences
6214 of the preceding ERE, `"{m,}"` matches at least *m* occurrences, and `"{m,n}"` matches any
6215 number of occurrences between *m* and *n*, inclusive.

6216 For example, in the string `"abababcccccd"` the ERE `"c{3}"` is matched by characters
6217 `'7'` to `'9'` and the ERE `"(ab){2,}"` is matched by characters one to six.

6218 The behavior of multiple adjacent duplication symbols (`'+'`, `'*'`, `'?'`, and intervals) produces
6219 undefined results.

6220 An ERE matching a single character repeated by an `'*'`, `'?'`, or an interval expression shall not
6221 match a null expression unless this is the only match for the repetition or it is necessary to satisfy
6222 the exact or minimum number of occurrences for the interval expression.

6223 **9.4.7 ERE Alternation**

6224 Two EREs separated by the special character vertical-line ('|') shall match a string that is
 6225 matched by either. For example, the ERE "a((bc)|d)" matches the string "abc" and the string
 6226 "ad". Single characters, or expressions matching single characters, separated by the vertical bar
 6227 and enclosed in parentheses, shall be treated as an ERE matching a single character.

6228 **9.4.8 ERE Precedence**

6229 The order of precedence shall be as shown in the following table:

ERE Precedence (from high to low)	
6230 Collation-related bracket symbols	[==] [::] [..]
6231 Escaped characters	\<special character>
6232 Bracket expression	[]
6233 Grouping	()
6234 Single-character-ERE duplication	* + ? {m,n}
6235 Concatenation	
6236 Anchoring	^ \$
6237 Alternation	

6239 For example, the ERE "abba|cde" matches either the string "abba" or the string "cde"
 6240 (rather than the string "abbade" or "abbcde", because concatenation has a higher order of
 6241 precedence than alternation).

6242 **9.4.9 ERE Expression Anchoring**

6243 An ERE can be limited to matching strings that begin or end a line; this is called *anchoring*. The
 6244 circumflex and dollar sign special characters shall be considered ERE anchors when used
 6245 anywhere outside a bracket expression. This shall have the following effects:

- 6246 1. A circumflex ('^') outside a bracket expression shall anchor the expression or
 6247 subexpression it begins to the beginning of a string; such an expression or subexpression
 6248 can match only a sequence starting at the first character of a string. For example, the EREs
 6249 "^ab" and "(^ab)" match "ab" in the string "abcdef", but fail to match in the string
 6250 "cdefab", and the ERE "a^b" is valid, but can never match because the 'a' prevents the
 6251 expression "^b" from matching starting at the first character.
- 6252 2. A dollar sign ('\$') outside a bracket expression shall anchor the expression or
 6253 subexpression it ends to the end of a string; such an expression or subexpression can
 6254 match only a sequence ending at the last character of a string. For example, the EREs
 6255 "ef\$" and "(ef\$)" match "ef" in the string "abcdef", but fail to match in the string
 6256 "cdefab", and the ERE "e\$f" is valid, but can never match because the 'f' prevents the
 6257 expression "e\$" from matching ending at the last character.

6258 **9.5 Regular Expression Grammar**

6259 Grammars describing the syntax of both basic and extended regular expressions are presented in
 6260 this section. The grammar takes precedence over the text. See the Shell and Utilities volume of
 6261 IEEE Std 1003.1-200x, Section 1.10, Grammar Conventions.

6262 **9.5.1 BRE/ERE Grammar Lexical Conventions**

6263 The lexical conventions for regular expressions are as described in this section.

6264 Except as noted, the longest possible token or delimiter beginning at a given point is recognized.

6265 The following tokens are processed (in addition to those string constants shown in the
 6266 grammar):

6267 COLL_ELEM_SINGLE

6268 Any single-character collating element, unless it is a META_CHAR.

6269 COLL_ELEM_MULTI Any multi-character collating element.

6270 BACKREF

6271 Applicable only to basic regular expressions. The character string
 consisting of '\ ' followed by a single-digit numeral, '1' to '9'.

6272 DUP_COUNT

6273 Represents a numeric constant. It shall be an integer in the range 0
 6274 ≤DUP_COUNT ≤{RE_DUP_MAX}. This token is only recognized when
 6275 the context of the grammar requires it. At all other times, digits not
 preceded by '\ ' are treated as ORD_CHAR.

6276 META_CHAR

One of the characters:

6277 ^ When found first in a bracket expression

6278 – When found anywhere but first (after an initial '^', if any) or
 6279 last in a bracket expression, or as the ending range point in a
 6280 range expression

6281] When found anywhere but first (after an initial '^', if any) in a
 6282 bracket expression

6283 L_ANCHOR

6284 Applicable only to basic regular expressions. The character '^' when it
 6285 appears as the first character of a basic regular expression and when not
 6286 QUOTED_CHAR. The '^' may be recognized as an anchor elsewhere;
 see Section 9.3.8 (on page 171).

6287 ORD_CHAR

A character, other than one of the special characters in SPEC_CHAR.

6288 QUOTED_CHAR

In a BRE, one of the character sequences:

6289 \^ \. * \[\ \$ \ \

6290 In an ERE, one of the character sequences:

6291 \^ \. \[\ \$ \ (\) \ |
 6292 * \+ \? \{ \ \

6293 R_ANCHOR

6294 (Applicable only to basic regular expressions.) The character '\$' when it
 6295 appears as the last character of a basic regular expression and when not
 6296 QUOTED_CHAR. The '\$' may be recognized as an anchor elsewhere;
 see Section 9.3.8 (on page 171).

6297 SPEC_CHAR

For basic regular expressions, one of the following special characters:

6298	.	Anywhere outside bracket expressions
6299	\	Anywhere outside bracket expressions
6300	[Anywhere outside bracket expressions
6301	^	When used as an anchor (see Section 9.3.8 (on page 171)) or
6302		when first in a bracket expression
6303	\$	When used as an anchor
6304	*	Anywhere except first in an entire RE, anywhere in a bracket
6305		expression, directly following "\(", directly following an
6306		anchoring '^'
6307		For extended regular expressions, shall be one of the following special
6308		characters found anywhere outside bracket expressions:
6309	^ . [\$ ()	
6310	* + ? { \	
6311		(The close-parenthesis shall be considered special in this context only if
6312		matched with a preceding open-parenthesis.)

6313 9.5.2 RE and Bracket Expression Grammar

6314 This section presents the grammar for basic regular expressions, including the bracket
6315 expression grammar that is common to both BREs and EREs.

```

6316 %token   ORD_CHAR QUOTED_CHAR DUP_COUNT
6317 %token   BACKREF L_ANCHOR R_ANCHOR
6318 %token   Back_open_paren  Back_close_paren
6319 /*      '\('           '\)'           */
6320 %token   Back_open_brace  Back_close_brace
6321 /*      '\{'           '\}'           */
6322 /* The following tokens are for the Bracket Expression
6323    grammar common to both REs and EREs. */
6324 %token   COLL_ELEM_SINGLE COLL_ELEM_MULTI META_CHAR
6325 %token   Open_equal Equal_close Open_dot Dot_close Open_colon Colon_close
6326 /*      '['           '='           '[' '.'           '.']'           '[' ':'           ':']' */
6327 %token   class_name
6328 /* class_name is a keyword to the LC_CTYPE locale category */
6329 /* (representing a character class) in the current locale */
6330 /* and is only recognized between [: and :] */
6331 %start   basic_reg_exp
6332 %%
6333 /* -----
6334    Basic Regular Expression
6335    -----
6336 */
6337 basic_reg_exp : RE_expression
6338               | L_ANCHOR
6339               | R_ANCHOR

```

```

6340         | L_ANCHOR                R_ANCHOR
6341         | L_ANCHOR RE_expression
6342         | RE_expression R_ANCHOR
6343         | L_ANCHOR RE_expression R_ANCHOR
6344         ;
6345 RE_expression : simple_RE
6346         | RE_expression simple_RE
6347         ;
6348 simple_RE : nondupl_RE
6349         | nondupl_RE RE_dupl_symbol
6350         ;
6351 nondupl_RE : one_char_or_coll_elem_RE
6352         | Back_open_paren RE_expression Back_close_paren
6353         | BACKREF
6354         ;
6355 one_char_or_coll_elem_RE : ORD_CHAR
6356         | QUOTED_CHAR
6357         | '.'
6358         | bracket_expression
6359         ;
6360 RE_dupl_symbol : '*'
6361         | Back_open_brace DUP_COUNT                Back_close_brace
6362         | Back_open_brace DUP_COUNT ','            Back_close_brace
6363         | Back_open_brace DUP_COUNT ',' DUP_COUNT Back_close_brace
6364         ;

6365 /* -----
6366    Bracket Expression
6367    -----
6368 */
6369 bracket_expression : '[' matching_list ']'
6370         | '[' nonmatching_list ']'
6371         ;
6372 matching_list : bracket_list
6373         ;
6374 nonmatching_list : '^' bracket_list
6375         ;
6376 bracket_list : follow_list
6377         | follow_list '-'
6378         ;
6379 follow_list : expression_term
6380         | follow_list expression_term
6381         ;
6382 expression_term : single_expression
6383         | range_expression
6384         ;
6385 single_expression : end_range
6386         | character_class
6387         | equivalence_class
6388         ;
6389 range_expression : start_range end_range
6390         | start_range '-'
6391         ;

```

```

6392     start_range      : end_range '-'
6393                       ;
6394     end_range         : COLL_ELEM_SINGLE
6395                       | collating_symbol
6396                       ;
6397     collating_symbol  : Open_dot COLL_ELEM_SINGLE Dot_close
6398                       | Open_dot COLL_ELEM_MULTI Dot_close
6399                       | Open_dot META_CHAR Dot_close
6400                       ;
6401     equivalence_class : Open_equal COLL_ELEM_SINGLE Equal_close
6402                       | Open_equal COLL_ELEM_MULTI Equal_close
6403                       ;
6404     character_class   : Open_colon class_name Colon_close
6405                       ;

```

6406 The BRE grammar does not permit L_ANCHOR or R_ANCHOR inside "\(" and "\)" (which
6407 implies that '^' and '\$' are ordinary characters). This reflects the semantic limits on the
6408 application, as noted in Section 9.3.8 (on page 171). Implementations are permitted to extend the
6409 language to interpret '^' and '\$' as anchors in these locations, and as such, conforming |
6410 applications cannot use unescaped '^' and '\$' in positions inside "\(" and "\)" that might |
6411 be interpreted as anchors.

6412 9.5.3 ERE Grammar

6413 This section presents the grammar for extended regular expressions, excluding the bracket
6414 expression grammar.

6415 **Note:** The bracket expression grammar and the associated %token lines are identical between BREs
6416 and EREs. It has been omitted from the ERE section to avoid unnecessary editorial duplication.

```

6417 %token  ORD_CHAR QUOTED_CHAR DUP_COUNT
6418 %start  extended_reg_exp
6419 %%
6420 /* -----
6421    Extended Regular Expression
6422    -----
6423 */
6424 extended_reg_exp  :                               ERE_branch
6425                   | extended_reg_exp '|' ERE_branch
6426                   ;
6427 ERE_branch        :                               ERE_expression
6428                   | ERE_branch ERE_expression
6429                   ;
6430 ERE_expression    : one_char_or_coll_elem_ERE
6431                   | '^'
6432                   | '$'
6433                   | '(' extended_reg_exp ')'
6434                   | ERE_expression ERE_dupl_symbol
6435                   ;
6436 one_char_or_coll_elem_ERE : ORD_CHAR
6437                           | QUOTED_CHAR
6438                           | '.'
6439                           | bracket_expression
6440                           ;

```

```

6441     ERE_dupl_symbol      : '*'
6442                           | '+'
6443                           | '?'
6444                           | '{' DUP_COUNT '}'
6445                           | '{' DUP_COUNT ',' '}'
6446                           | '{' DUP_COUNT ',' DUP_COUNT '}'
6447                           ;

```

6448 The ERE grammar does not permit several constructs that previous sections specify as having
 6449 undefined results:

- 6450 • ORD_CHAR preceded by '\'
- 6451 • One or more *ERE_dupl_symbols* appearing first in an ERE, or immediately following '|',
 6452 '^', or '('
- 6453 • '{' not part of a valid *ERE_dupl_symbol*
- 6454 • '|' appearing first or last in an ERE, or immediately following '|' or '(', or immediately
 6455 preceding ')'

6456 Implementations are permitted to extend the language to allow these. Conforming applications |
 6457 cannot use such constructs. |

Directory Structure and Devices

6459

6460 10.1 Directory Structure and Files

6461 The following directories shall exist on conforming systems and conforming applications shall |
 6462 make use of them only as described. Strictly conforming applications shall not assume the |
 6463 ability to create files in any of these directories, unless specified below.

6464 / The root directory.

6465 /dev Contains /dev/console, /dev/null, and /dev/tty, described below.

6466 The following directory shall exist on conforming systems and shall be used as described.

6467 /tmp A directory made available for programs that need a place to create temporary |
 6468 files. Applications shall be allowed to create files in this directory, but shall not |
 6469 assume that such files are preserved between invocations of the application.

6470 The following files shall exist on conforming systems and shall be both readable and writable.

6471 /dev/null An infinite data source and data sink. Data written to /dev/null shall be discarded.
 6472 Reads from /dev/null shall always return end-of-file (EOF).

6473 /dev/tty In each process, a synonym for the controlling terminal associated with the process
 6474 group of that process, if any. It is useful for programs or shell procedures that wish
 6475 to be sure of writing messages to or reading data from the terminal no matter how
 6476 output has been redirected. It can also be used for programs that demand the name
 6477 of a file for output, when typed output is desired and it is tiresome to find out
 6478 what terminal is currently in use.

6479 The following file shall exist on conforming systems and need not be readable or writable:

6480 /dev/console The /dev/console file is a generic name given to the system console (see Section |
 6481 3.382 (on page 85)). It is usually linked to an implementation-defined special file. It |
 6482 shall provide an interface to the system console conforming to the requirements of |
 6483 the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal |
 6484 Interface. |

6485 10.2 Output Devices and Terminal Types

6486 The utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x historically have been
 6487 implemented on a wide range of terminal types, but a conforming implementation need not
 6488 support all features of all utilities on every conceivable terminal. IEEE Std 1003.1-200x states
 6489 which features are optional for certain classes of terminals in the individual utility description
 6490 sections. The implementation shall document which terminal types it supports and which of
 6491 these features and utilities are not supported by each terminal.

6492 When a feature or utility is not supported on a specific terminal type, as allowed by
 6493 IEEE Std 1003.1-200x, and the implementation considers such a condition to be an error
 6494 preventing use of the feature or utility, the implementation shall indicate such conditions
 6495 through diagnostic messages or exit status values or both (as appropriate to the specific utility
 6496 description) that inform the user that the terminal type lacks the appropriate capability.

6497 IEEE Std 1003.1-200x uses a notational convention based on historical practice that identifies
 6498 some of the control characters defined in Section 7.3.1 (on page 122) in a manner easily
 6499 remembered by users on many terminals. The correspondence between this “<control>-char”
 6500 notation and the actual control characters is shown in the following table. When
 6501 IEEE Std 1003.1-200x refers to a character by its <control>- name, it is referring to the actual
 6502 control character shown in the Value column of the table, which is not necessarily the exact
 6503 control key sequence on all terminals. Some terminals have keyboards that do not allow the
 6504 direct transmission of all the non-alphanumeric characters shown. In such cases, the system
 6505 documentation shall describe which data sequences transmitted by the terminal are interpreted
 6506 by the system as representing the special characters.

6507 **Table 10-1** Control Character Names

Name	Value	Symbolic Name	Name	Value	Symbolic Name
<control>-A	<SOH>	<SOH>	<control>-Q	<DC1>	<DC1>
<control>-B	<STX>	<STX>	<control>-R	<DC2>	<DC2>
<control>-C	<ETX>	<ETX>	<control>-S	<DC3>	<DC3>
<control>-D	<EOT>	<EOT>	<control>-T	<DC4>	<DC4>
<control>-E	<ENQ>	<ENQ>	<control>-U	<NAK>	<NAK>
<control>-F	<ACK>	<ACK>	<control>-V	<SYN>	<SYN>
<control>-G	<BEL>	<alert>	<control>-W	<ETB>	<ETB>
<control>-H	<BS>	<backspace>	<control>-X	<CAN>	<CAN>
<control>-I	<HT>	<tab>	<control>-Y		
<control>-J	<LF>	<linefeed>	<control>-Z	<SUB>	<SUB>
<control>-K	<VT>	<vertical-tab>	<control>-[<ESC>	<ESC>
<control>-L	<FF>	<form-feed>	<control>-\	<FS>	<FS>
<control>-M	<CR>	<carriage-return>	<control>-]	<GS>	<GS>
<control>-N	<SO>	<SO>	<control>-^	<RS>	<RS>
<control>-O	<SI>	<SI>	<control>-_	<US>	<US>
<control>-P	<DLE>	<DLE>	<control>-?		

6525 **Note:** The notation uses uppercase letters for arbitrary editorial reasons. There is no implication that
 6526 the keystrokes represent control-shift-letter sequences.

General Terminal Interface

6527

6528 This chapter describes a general terminal interface that shall be provided. It shall be supported
6529 on any asynchronous communications ports if the implementation provides them. It is
6530 implementation-defined whether it supports network connections or synchronous ports, or
6531 both.

6532 11.1 Interface Characteristics

6533 11.1.1 Opening a Terminal Device File

6534 When a terminal device file is opened, it normally causes the thread to wait until a connection is
6535 established. In practice, application programs seldom open these files; they are opened by
6536 special programs and become an application's standard input, output, and error files.

6537 As described in *open()*, opening a terminal device file with the *O_NONBLOCK* flag clear shall
6538 cause the thread to block until the terminal device is ready and available. If *CLOCAL* mode is
6539 not set, this means blocking until a connection is established. If *CLOCAL* mode is set in the
6540 terminal, or the *O_NONBLOCK* flag is specified in the *open()*, the *open()* function shall return a
6541 file descriptor without waiting for a connection to be established.

6542 11.1.2 Process Groups

6543 A terminal may have a foreground process group associated with it. This foreground process
6544 group plays a special role in handling signal-generating input characters, as discussed in Section
6545 11.1.9 (on page 187).

6546 A command interpreter process supporting job control can allocate the terminal to different jobs,
6547 or process groups, by placing related processes in a single process group and associating this
6548 process group with the terminal. A terminal's foreground process group may be set or examined
6549 by a process, assuming the permission requirements are met; see *tcgetpgrp()* and *tcsetpgrp()*. The
6550 terminal interface aids in this allocation by restricting access to the terminal by processes that are
6551 not in the current process group; see Section 11.1.4 (on page 184).

6552 When there is no longer any process whose process ID or process group ID matches the process
6553 group ID of the foreground process group, the terminal shall have no foreground process group.
6554 It is unspecified whether the terminal has a foreground process group when there is a process
6555 whose process ID matches the foreground process ID, but whose process group ID does not. No
6556 actions defined in IEEE Std 1003.1-200x, other than allocation of a controlling terminal or a
6557 successful call to *tcsetpgrp()*, cause a process group to become the foreground process group of
6558 the terminal.

6559 11.1.3 The Controlling Terminal

6560 A terminal may belong to a process as its controlling terminal. Each process of a session that has
6561 a controlling terminal has the same controlling terminal. A terminal may be the controlling
6562 terminal for at most one session. The controlling terminal for a session is allocated by the session
6563 leader in an implementation-defined manner. If a session leader has no controlling terminal, and
6564 opens a terminal device file that is not already associated with a session without using the
6565 O_NOCTTY option (see *open()*), it is implementation-defined whether the terminal becomes the
6566 controlling terminal of the session leader. If a process which is not a session leader opens a
6567 terminal file, or the O_NOCTTY option is used on *open()*, then that terminal shall not become
6568 the controlling terminal of the calling process. When a controlling terminal becomes associated
6569 with a session, its foreground process group shall be set to the process group of the session
6570 leader.

6571 The controlling terminal is inherited by a child process during a *fork()* function call. A process
6572 relinquishes its controlling terminal when it creates a new session with the *setsid()* function;
6573 other processes remaining in the old session that had this terminal as their controlling terminal
6574 continue to have it. Upon the close of the last file descriptor in the system (whether or not it is in
6575 the current session) associated with the controlling terminal, it is unspecified whether all
6576 processes that had that terminal as their controlling terminal cease to have any controlling
6577 terminal. Whether and how a session leader can reacquire a controlling terminal after the
6578 controlling terminal has been relinquished in this fashion is unspecified. A process does not
6579 relinquish its controlling terminal simply by closing all of its file descriptors associated with the
6580 controlling terminal if other processes continue to have it open.

6581 When a controlling process terminates, the controlling terminal is dissociated from the current
6582 session, allowing it to be acquired by a new session leader. Subsequent access to the terminal by
6583 other processes in the earlier session may be denied, with attempts to access the terminal treated
6584 as if a modem disconnect had been sensed.

6585 11.1.4 Terminal Access Control

6586 If a process is in the foreground process group of its controlling terminal, read operations shall
6587 be allowed, as described in Section 11.1.5 (on page 185). Any attempts by a process in a
6588 background process group to read from its controlling terminal cause its process group to be
6589 sent a SIGTTIN signal unless one of the following special cases applies: if the reading process is
6590 ignoring or blocking the SIGTTIN signal, or if the process group of the reading process is
6591 orphaned, the *read()* shall return -1 , with *errno* set to [EIO] and no signal shall be sent. The
6592 default action of the SIGTTIN signal shall be to stop the process to which it is sent. See
6593 **<signal.h>**.

6594 If a process is in the foreground process group of its controlling terminal, write operations shall
6595 be allowed as described in Section 11.1.8 (on page 187). Attempts by a process in a background
6596 process group to write to its controlling terminal shall cause the process group to be sent a
6597 SIGTTOU signal unless one of the following special cases applies: if TOSTOP is not set, or if
6598 TOSTOP is set and the process is ignoring or blocking the SIGTTOU signal, the process is
6599 allowed to write to the terminal and the SIGTTOU signal is not sent. If TOSTOP is set, and the
6600 process group of the writing process is orphaned, and the writing process is not ignoring or
6601 blocking the SIGTTOU signal, the *write()* shall return -1 , with *errno* set to [EIO] and no signal
6602 shall be sent.

6603 Certain calls that set terminal parameters are treated in the same fashion as *write()*, except that
6604 TOSTOP is ignored; that is, the effect is identical to that of terminal writes when TOSTOP is set
6605 (see Section 11.2.5 (on page 193), *tcdrain()*, *tcfLOW()*, *tcfLush()*, *tcsendbreak()*, *tcsetattr()*, and
6606 *tcsetpgrp()*).

6607 11.1.5 Input Processing and Reading Data

6608 A terminal device associated with a terminal device file may operate in full-duplex mode, so that
 6609 data may arrive even while output is occurring. Each terminal device file has an *input queue*,
 6610 associated with it, into which incoming data is stored by the system before being read by a
 6611 process. The system may impose a limit, {MAX_INPUT}, on the number of bytes that may be
 6612 stored in the input queue. The behavior of the system when this limit is exceeded is
 6613 implementation-defined.

6614 Two general kinds of input processing are available, determined by whether the terminal device
 6615 file is in canonical mode or non-canonical mode. These modes are described in Section 11.1.6 and
 6616 Section 11.1.7 (on page 186). Additionally, input characters are processed according to the
 6617 *c_iflag* (see Section 11.2.2 (on page 189)) and *c_lflag* (see Section 11.2.5 (on page 193)) fields.
 6618 Such processing can include *echoing*, which in general means transmitting input characters
 6619 immediately back to the terminal when they are received from the terminal. This is useful for
 6620 terminals that can operate in full-duplex mode.

6621 The manner in which data is provided to a process reading from a terminal device file is
 6622 dependent on whether the terminal file is in canonical or non-canonical mode, and on whether
 6623 or not the O_NONBLOCK flag is set by *open()* or *fcntl()*.

6624 If the O_NONBLOCK flag is clear, then the read request shall be blocked until data is available
 6625 or a signal has been received. If the O_NONBLOCK flag is set, then the read request shall be
 6626 completed, without blocking, in one of three ways:

- 6627 1. If there is enough data available to satisfy the entire request, the *read()* shall complete
 6628 successfully and shall return the number of bytes read.
- 6629 2. If there is not enough data available to satisfy the entire request, the *read()* shall complete
 6630 successfully, having read as much data as possible, and shall return the number of bytes it
 6631 was able to read.
- 6632 3. If there is no data available, the *read()* shall return -1 , with *errno* set to [EAGAIN].

6633 When data is available depends on whether the input processing mode is canonical or non-
 6634 canonical. The following sections, Section 11.1.6 and Section 11.1.7 (on page 186), describe each
 6635 of these input processing modes.

6636 11.1.6 Canonical Mode Input Processing

6637 In canonical mode input processing, terminal input is processed in units of lines. A line is
 6638 delimited by a newline character (NL), an end-of-file character (EOF), or an end-of-line (EOL)
 6639 character. See Section 11.1.9 (on page 187) for more information on EOF and EOL. This means
 6640 that a read request shall not return until an entire line has been typed or a signal has been
 6641 received. Also, no matter how many bytes are requested in the *read()* call, at most one line shall
 6642 be returned. It is not, however, necessary to read a whole line at once; any number of bytes, even
 6643 one, may be requested in a *read()* without losing information.

6644 If {MAX_CANON} is defined for this terminal device, it shall be a limit on the number of bytes |
 6645 in a line. The behavior of the system when this limit is exceeded is implementation-defined. If |
 6646 {MAX_CANON} is not defined, there shall be no such limit; see *pathconf()*. |

6647 Erase and kill processing occur when either of two special characters, the ERASE and KILL |
 6648 characters (see Section 11.1.9 (on page 187)), is received. This processing shall affect data in the |
 6649 input queue that has not yet been delimited by a newline (NL), EOF, or EOL character. This un- |
 6650 delimited data makes up the current line. The ERASE character shall delete the last character in |
 6651 the current line, if there is one. The KILL character shall delete all data in the current line, if there |
 6652 are any. The ERASE and KILL characters shall have no effect if there is no data in the current |

6653 line. The ERASE and KILL characters themselves shall not be placed in the input queue. |

6654 11.1.7 Non-Canonical Mode Input Processing

6655 In non-canonical mode input processing, input bytes are not assembled into lines, and erase and |
6656 kill processing shall not occur. The values of the MIN and TIME members of the `c_cc` array are |
6657 used to determine how to process the bytes received. The IEEE Std 1003.1-200x does not specify |
6658 whether the setting of `O_NONBLOCK` takes precedence over MIN or TIME settings. Therefore, |
6659 if `O_NONBLOCK` is set, `read()` may return immediately, regardless of the setting of MIN or |
6660 TIME. Also, if no data is available, `read()` may either return 0, or return `-1` with `errno` set to |
6661 `[EAGAIN]`.

6662 MIN represents the minimum number of bytes that should be received when the `read()` function |
6663 returns successfully. TIME is a timer of 0.1 second granularity that is used to time out bursty and |
6664 short-term data transmissions. If MIN is greater than `{MAX_INPUT}`, the response to the request |
6665 is undefined. The four possible values for MIN and TIME and their interactions are described |
6666 below.

6667 Case A: MIN>0, TIME>0

6668 In case A, TIME serves as an inter-byte timer which shall be activated after the first byte is |
6669 received. Since it is an inter-byte timer, it shall be reset after a byte is received. The interaction |
6670 between MIN and TIME is as follows. As soon as one byte is received, the inter-byte timer shall |
6671 be started. If MIN bytes are received before the inter-byte timer expires (remember that the timer |
6672 is reset upon receipt of each byte), the read shall be satisfied. If the timer expires before MIN |
6673 bytes are received, the characters received to that point shall be returned to the user. Note that if |
6674 TIME expires at least one byte shall be returned because the timer would not have been enabled |
6675 unless a byte was received. In this case (MIN>0, TIME>0) the read shall block until the MIN and |
6676 TIME mechanisms are activated by the receipt of the first byte, or a signal is received. If data is in |
6677 the buffer at the time of the `read()`, the result shall be as if data has been received immediately |
6678 after the `read()`.

6679 Case B: MIN>0, TIME=0

6680 In case B, since the value of TIME is zero, the timer plays no role and only MIN is significant. A |
6681 pending read shall not be satisfied until MIN bytes are received (that is, the pending read shall |
6682 block until MIN bytes are received), or a signal is received. A program that uses case B to read |
6683 record-based terminal I/O may block indefinitely in the read operation.

6684 Case C: MIN=0, TIME>0

6685 In case C, since MIN=0, TIME no longer represents an inter-byte timer. It now serves as a read |
6686 timer that shall be activated as soon as the `read()` function is processed. A read shall be satisfied |
6687 as soon as a single byte is received or the read timer expires. Note that in case C if the timer |
6688 expires, no bytes shall be returned. If the timer does not expire, the only way the read can be |
6689 satisfied is if a byte is received. If bytes are not received, the read shall not block indefinitely |
6690 waiting for a byte; if no byte is received within `TIME*0.1` seconds after the read is initiated, the |
6691 `read()` shall return a value of zero, having read no data. If data is in the buffer at the time of the |
6692 `read()`, the timer shall be started as if data has been received immediately after the `read()`.

6693 **Case D: MIN=0, TIME=0**

6694 The minimum of either the number of bytes requested or the number of bytes currently available
 6695 shall be returned without waiting for more bytes to be input. If no characters are available, *read()*
 6696 shall return a value of zero, having read no data.

6697 **11.1.8 Writing Data and Output Processing**

6698 When a process writes one or more bytes to a terminal device file, they are processed according
 6699 to the *c_oflag* field (see Section 11.2.3 (on page 190)). The implementation may provide a
 6700 buffering mechanism; as such, when a call to *write()* completes, all of the bytes written have
 6701 been scheduled for transmission to the device, but the transmission has not necessarily
 6702 completed. See *write()* for the effects of *O_NONBLOCK* on *write()*.

6703 **11.1.9 Special Characters**

6704 Certain characters have special functions on input or output or both. These functions are
 6705 summarized as follows:

6706 **INTR** Special character on input, which is recognized if the *ISIG* flag is set. Generates a
 6707 *SIGINT* signal which is sent to all processes in the foreground process group for which
 6708 the terminal is the controlling terminal. If *ISIG* is set, the *INTR* character shall be
 6709 discarded when processed. |

6710 **QUIT** Special character on input, which is recognized if the *ISIG* flag is set. Generates a
 6711 *SIGQUIT* signal which is sent to all processes in the foreground process group for
 6712 which the terminal is the controlling terminal. If *ISIG* is set, the *QUIT* character shall be
 6713 discarded when processed. |

6714 **ERASE** Special character on input, which is recognized if the *ICANON* flag is set. Erases the
 6715 last character in the current line; see Section 11.1.6 (on page 185). It shall not erase
 6716 beyond the start of a line, as delimited by an *NL*, *EOF*, or *EOL* character. If *ICANON* is
 6717 set, the *ERASE* character shall be discarded when processed. |

6718 **KILL** Special character on input, which is recognized if the *ICANON* flag is set. Deletes the
 6719 entire line, as delimited by an *NL*, *EOF*, or *EOL* character. If *ICANON* is set, the *KILL*
 6720 character shall be discarded when processed. |

6721 **EOF** Special character on input, which is recognized if the *ICANON* flag is set. When
 6722 received, all the bytes waiting to be read are immediately passed to the process without
 6723 waiting for a newline, and the *EOF* is discarded. Thus, if there are no bytes waiting
 6724 (that is, the *EOF* occurred at the beginning of a line), a byte count of zero shall be
 6725 returned from the *read()*, representing an end-of-file indication. If *ICANON* is set, the
 6726 *EOF* character shall be discarded when processed. |

6727 **NL** Special character on input, which is recognized if the *ICANON* flag is set. It is the line
 6728 delimiter newline. It cannot be changed.

6729 **EOL** Special character on input, which is recognized if the *ICANON* flag is set. It is an
 6730 additional line delimiter, like *NL*.

6731 **SUSP** If the *ISIG* flag is set, receipt of the *SUSP* character shall cause a *SIGTSTP* signal to be
 6732 sent to all processes in the foreground process group for which the terminal is the
 6733 controlling terminal, and the *SUSP* character shall be discarded when processed. |

6734 **STOP** Special character on both input and output, which is recognized if the *IXON* (output
 6735 control) or *IXOFF* (input control) flag is set. Can be used to suspend output
 6736 temporarily. It is useful with CRT terminals to prevent output from disappearing

6737 before it can be read. If IXON is set, the STOP character shall be discarded when
6738 processed.

6739 **START** Special character on both input and output, which is recognized if the IXON (output
6740 control) or IXOFF (input control) flag is set. Can be used to resume output that has
6741 been suspended by a STOP character. If IXON is set, the START character shall be
6742 discarded when processed.

6743 **CR** Special character on input, which is recognized if the ICANON flag is set; it is the
6744 carriage-return character. When ICANON and ICRNL are set and IGNCR is not set,
6745 this character shall be translated into an NL, and shall have the same effect as an NL
6746 character.

6747 The NL and CR characters cannot be changed. It is implementation-defined whether the START
6748 and STOP characters can be changed. The values for INTR, QUIT, ERASE, KILL, EOF, EOL, and
6749 SUSP shall be changeable to suit individual tastes. Special character functions associated with
6750 changeable special control characters can be disabled individually.

6751 If two or more special characters have the same value, the function performed when that
6752 character is received is undefined.

6753 A special character is recognized not only by its value, but also by its context; for example, an
6754 implementation may support multi-byte sequences that have a meaning different from the
6755 meaning of the bytes when considered individually. Implementations may also support
6756 additional single-byte functions. These implementation-defined multi-byte or single-byte
6757 functions shall be recognized only if the IEXTEN flag is set; otherwise, data is received without
6758 interpretation, except as required to recognize the special characters defined in this section.

6759 **XSI** If IEXTEN is set, the ERASE, KILL, and EOF characters can be escaped by a preceding '\'
6760 character, in which case no special function shall occur.

6761 **11.1.10 Modem Disconnect**

6762 If a modem disconnect is detected by the terminal interface for a controlling terminal, and if
6763 CLOCAL is not set in the **c_cflag** field for the terminal (see Section 11.2.4 (on page 192)), the
6764 SIGHUP signal shall be sent to the controlling process for which the terminal is the controlling
6765 terminal. Unless other arrangements have been made, this shall cause the controlling process to
6766 terminate (see *exit()*). Any subsequent read from the terminal device shall return the value of
6767 zero, indicating end-of-file; see *read()*. Thus, processes that read a terminal file and test for end-
6768 of-file can terminate appropriately after a disconnect. If the EIO condition as specified in *read()*
6769 also exists, it is unspecified whether on EOF condition or the [EIO] is returned. Any subsequent
6770 *write()* to the terminal device shall return -1 , with *errno* set to [EIO], until the device is closed.

6771 **11.1.11 Closing a Terminal Device File**

6772 The last process to close a terminal device file shall cause any output to be sent to the device and
6773 any input to be discarded. If HUPCL is set in the control modes and the communications port
6774 supports a disconnect function, the terminal device shall perform a disconnect.

6775 **11.2 Parameters that Can be Set**

6776 **11.2.1 The termios Structure**

6777 Routines that need to control certain terminal I/O characteristics shall do so by using the
 6778 **termios** structure as defined in the `<termios.h>` header. The members of this structure include
 6779 (but are not limited to):

Member Type	Array Size	Member Name	Description
6780 <code>tflag_t</code>		<code>c_iflag</code>	Input modes.
6781 <code>tflag_t</code>		<code>c_oflag</code>	Output modes.
6782 <code>tflag_t</code>		<code>c_cflag</code>	Control modes.
6783 <code>tflag_t</code>		<code>c_lflag</code>	Local modes.
6784 <code>cc_t</code>	NCCS	<code>c_cc[]</code>	Control characters.

6787 The types `tflag_t` and `cc_t` are defined in the `<termios.h>` header. They shall be unsigned
 6788 integer types.

6789 **11.2.2 Input Modes**

6790 Values of the `c_iflag` field describe the basic terminal input control, and are composed of the
 6791 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name
 6792 symbols in this table are defined in `<termios.h>`:

Mask Name	Description
6795 BRKINT	Signal interrupt on break.
6796 ICRNL	Map CR to NL on input.
6797 IGNBRK	Ignore break condition.
6798 IGNCR	Ignore CR.
6799 IGNPAR	Ignore characters with parity errors.
6800 INLCR	Map NL to CR on input.
6801 INPCK	Enable input parity check.
6802 ISTRIP	Strip character.
6803 XSI <code>IXANY</code>	Enable any character to restart output.
6804 <code>IXOFF</code>	Enable start/stop input control.
6805 <code>IXON</code>	Enable start/stop output control.
6806 <code>PARMRK</code>	Mark parity errors.

6807 In the context of asynchronous serial data transmission, a break condition shall be defined as a
 6808 sequence of zero-valued bits that continues for more than the time to send one byte. The entire
 6809 sequence of zero-valued bits is interpreted as a single break condition, even if it continues for a
 6810 time equivalent to more than one byte. In contexts other than asynchronous serial data
 6811 transmission, the definition of a break condition is implementation-defined.

6812 If IGNBRK is set, a break condition detected on input shall be ignored; that is, not put on the
 6813 input queue and therefore not read by any process. If IGNBRK is not set and BRKINT is set, the
 6814 break condition shall flush the input and output queues, and if the terminal is the controlling
 6815 terminal of a foreground process group, the break condition shall generate a single SIGINT
 6816 signal to that foreground process group. If neither IGNBRK nor BRKINT is set, a break
 6817 condition shall be read as a single 0x00, or if PARMRK is set, as 0xff 0x00 0x00.

6818 If IGNPAR is set, a byte with a framing or parity error (other than break) shall be ignored.

6819 If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than |
6820 break) shall be given to the application as the three-byte sequence 0xff 0x00 X, where 0xff 0x00 is |
6821 a two-byte flag preceding each sequence and X is the data of the byte received in error. To avoid |
6822 ambiguity in this case, if ISTRIP is not set, a valid byte of 0xff is given to the application as 0xff |
6823 0xff. If neither PARMRK nor IGNPAR is set, a framing or parity error (other than break) shall be |
6824 given to the application as a single byte 0x00. |

6825 If INPCK is set, input parity checking shall be enabled. If INPCK is not set, input parity checking |
6826 shall be disabled, allowing output parity generation without input parity errors. Note that |
6827 whether input parity checking is enabled or disabled is independent of whether parity detection |
6828 is enabled or disabled (see Section 11.2.4 (on page 192)). If parity detection is enabled but input |
6829 parity checking is disabled, the hardware to which the terminal is connected shall recognize the |
6830 parity bit, but the terminal special file shall not check whether or not this bit is correctly set.

6831 If ISTRIP is set, valid input bytes shall first be stripped to seven bits; otherwise, all eight bits |
6832 shall be processed.

6833 If INLCR is set, a received NL character shall be translated into a CR character. If IGNCR is set, a |
6834 received CR character shall be ignored (not read). If IGNCR is not set and ICRNL is set, a |
6835 received CR character shall be translated into an NL character.

6836 XSI If IXANY is set, any input character shall restart output that has been suspended.

6837 If IXON is set, start/stop output control shall be enabled. A received STOP character shall |
6838 suspend output and a received START character shall restart output. When IXON is set, START |
6839 and STOP characters are not read, but merely perform flow control functions. When IXON is not |
6840 set, the START and STOP characters shall be read.

6841 If IXOFF is set, start/stop input control shall be enabled. The system shall transmit STOP |
6842 characters, which are intended to cause the terminal device to stop transmitting data, as needed |
6843 to prevent the input queue from overflowing and causing implementation-defined behavior, and |
6844 shall transmit START characters, which are intended to cause the terminal device to resume |
6845 transmitting data, as soon as the device can continue transmitting data without risk of |
6846 overflowing the input queue. The precise conditions under which STOP and START characters |
6847 are transmitted are implementation-defined.

6848 The initial input control value after *open()* is implementation-defined.

6849 11.2.3 Output Modes

6850 The **c_oflag** field specifies the terminal interface's treatment of output, and is composed of the |
6851 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name |
6852 symbols in this table are defined in **<termios.h>**:

6853
6854
6855
6856
6857
6858
6859
6860
6861
6862
6863
6864
6865
6866
6867
6868
6869
6870
6871
6872
6873
6874
6875
6876
6877
6878
6879
6880
6881
6882
6883

Mask Name	Description
OPOST	Perform output processing.
ONLCR	Map NL to CR-NL on output.
OCRNL	Map CR to NL on output.
ONOCR	No CR output at column 0.
ONLRET	NL performs CR function.
OFILL	Use fill characters for delay.
OFDEL	Fill is DEL, else NUL.
NLDLY	Select newline delays:
NL0	Newline character type 0.
NL1	Newline character type 1.
CRDLY	Select carriage-return delays:
CR0	Carriage-return delay type 0.
CR1	Carriage-return delay type 1.
CR2	Carriage-return delay type 2.
CR3	Carriage-return delay type 3.
TABDLY	Select horizontal-tab delays:
TAB0	Horizontal-tab delay type 0.
TAB1	Horizontal-tab delay type 1.
TAB2	Horizontal-tab delay type 2.
TAB3	Expand tabs to spaces.
BSDLY	Select backspace delays:
BS0	Backspace-delay type 0.
BS1	Backspace-delay type 1.
VTDLY	Select vertical-tab delays:
VT0	Vertical-tab delay type 0.
VT1	Vertical-tab delay type 1.
FFDLY	Select form-feed delays:
FF0	Form-feed delay type 0.
FF1	Form-feed delay type 1.

6884
6885
6886

6887
6888
6889
6890
6891
6892
6893

6894
6895
6896
6897
6898

6899

6900
6901

If OPOST is set, output data shall be post-processed as described below, so that lines of text are modified to appear appropriately on the terminal device; otherwise, characters shall be transmitted without change.

XSI

If ONLCR is set, the NL character shall be transmitted as the CR-NL character pair. If OCRNL is set, the CR character shall be transmitted as the NL character. If ONOCR is set, no CR character shall be transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer shall be set to 0 and the delays specified for CR shall be used. Otherwise, the NL character is assumed to do just the line-feed function; the column pointer remains unchanged. The column pointer shall also be set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 shall indicate no delay. If OFILL is set, fill characters shall be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character shall be DEL; otherwise, NUL.

If a form-feed or vertical-tab delay is specified, it shall last for about 2 seconds.

New-line delay shall last about 0.10 seconds. If ONLRET is set, the carriage-return delays shall be used instead of the newline delays. If OFILL is set, two fill characters shall be transmitted.

6902 Carriage-return delay type 1 shall be dependent on the current column position, type 2 shall be
 6903 about 0.10 seconds, and type 3 shall be about 0.15 seconds. If OFILL is set, delay type 1 shall
 6904 transmit two fill characters, and type 2, four fill characters.

6905 Horizontal-tab delay type 1 shall be dependent on the current column position. Type 2 shall be
 6906 about 0.10 seconds. Type 3 specifies that tabs shall be expanded into spaces. If OFILL is set, two
 6907 fill characters shall be transmitted for any delay.

6908 Backspace delay shall last about 0.05 seconds. If OFILL is set, one fill character shall be
 6909 transmitted.

6910 The actual delays depend on line speed and system load.

6911 The initial output control value after *open()* is implementation-defined.

6912 11.2.4 Control Modes

6913 The **c_cflag** field describes the hardware control of the terminal, and is composed of the
 6914 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name
 6915 symbols in this table are defined in **<termios.h>**; not all values specified are required to be
 6916 supported by the underlying hardware:

Mask Name	Description
CLOCAL	Ignore modem status lines.
CREAD	Enable receiver.
CSIZE	Number of bits transmitted or received per byte:
CS5	5 bits
CS6	6 bits
CS7	7 bits
CS8	8 bits.
CSTOPB	Send two stop bits, else one.
HUPCL	Hang up on last close.
PARENB	Parity enable.
PARODD	Odd parity, else even.

6929 In addition, the input and output baud rates are stored in the **termios** structure. The symbols in
 6930 the following table are defined in **<termios.h>**. Not all values specified are required to be
 6931 supported by the underlying hardware.

Name	Description	Name	Description
B0	Hang up	B600	600 baud
B50	50 baud	B1200	1200 baud
B75	75 baud	B1800	1800 baud
B110	110 baud	B2400	2400 baud
B134	134.5 baud	B4800	4800 baud
B150	150 baud	B9600	9600 baud
B200	200 baud	B19200	19200 baud
B300	300 baud	B38400	38400 baud

6941 The following functions are provided for getting and setting the values of the input and output
 6942 baud rates in the **termios** structure: *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, and *cfsetospeed()*.
 6943 The effects on the terminal device shall not become effective and not all errors need be detected
 6944 until the *tcsetattr()* function is successfully called.

6945 The CSIZE bits shall specify the number of transmitted or received bits per byte. If ISTRIP is not
 6946 set, the value of all the other bits is unspecified. If ISTRIP is set, the value of all but the 7 low-

6947 order bits shall be zero, but the value of any other bits beyond CSIZE is unspecified when read. |
 6948 CSIZE shall not include the parity bit, if any. If CSTOPB is set, two stop bits shall be used; |
 6949 otherwise, one stop bit. For example, at 110 baud, two stop bits are normally used. |

6950 If CREAD is set, the receiver shall be enabled; otherwise, no characters shall be received. |

6951 If PARENB is set, parity generation and detection shall be enabled and a parity bit is added to |
 6952 each byte. If parity is enabled, PARODD shall specify odd parity if set; otherwise, even parity |
 6953 shall be used. |

6954 If HUPCL is set, the modem control lines for the port shall be lowered when the last process |
 6955 with the port open closes the port or the process terminates. The modem connection shall be |
 6956 broken. |

6957 If CLOCAL is set, a connection shall not depend on the state of the modem status lines. If |
 6958 CLOCAL is clear, the modem status lines shall be monitored. |

6959 Under normal circumstances, a call to the *open()* function shall wait for the modem connection |
 6960 to complete. However, if the O_NONBLOCK flag is set (see *open()*) or if CLOCAL has been set, |
 6961 the *open()* function shall return immediately without waiting for the connection. |

6962 If the object for which the control modes are set is not an asynchronous serial connection, some |
 6963 of the modes may be ignored; for example, if an attempt is made to set the baud rate on a |
 6964 network connection to a terminal on another host, the baud rate need not be set on the |
 6965 connection between that terminal and the machine to which it is directly connected. |

6966 The initial hardware control value after *open()* is implementation-defined. |

6967 11.2.5 Local Modes

6968 The *c_lflag* field of the argument structure is used to control various functions. It is composed |
 6969 of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name |
 6970 symbols in this table are defined in `<termios.h>`; not all values specified are required to be |
 6971 supported by the underlying hardware: |

6972

6973

Mask Name	Description
ECHO	Enable echo.
ECHOE	Echo ERASE as an error correcting backspace.
ECHOK	Echo KILL.
ECHONL	Echo <newline>.
ICANON	Canonical input (erase and kill processing).
IEXTEN	Enable extended (implementation-defined) functions.
ISIG	Enable signals.
NOFLSH	Disable flush after interrupt, quit or suspend.
TOSTOP	Send SIGTTOU for background output.

6974

6975

6976

6977

6978

6979

6980

6981

6982

6983 If ECHO is set, input characters shall be echoed back to the terminal. If ECHO is clear, input |
 6984 characters shall not be echoed. |

6985 If ECHOE and ICANON are set, the ERASE character shall cause the terminal to erase, if |
 6986 possible, the last character in the current line from the display. If there is no character to erase, an |
 6987 implementation may echo an indication that this was the case, or do nothing. |

6988 If ECHOK and ICANON are set, the KILL character shall either cause the terminal to erase the |
 6989 line from the display or shall echo the newline character after the KILL character. |

6990 If ECHONL and ICANON are set, the newline character shall be echoed even if ECHO is not set.

6991 If ICANON is set, canonical processing shall be enabled. This enables the erase and kill edit
6992 functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL, as
6993 described in Section 11.1.6 (on page 185).

6994 If ICANON is not set, read requests shall be satisfied directly from the input queue. A read shall
6995 not be satisfied until at least MIN bytes have been received or the timeout value TIME expired
6996 between bytes. The time value represents tenths of a second. See Section 11.1.7 (on page 186) for
6997 more details.

6998 If IEXTEN is set, implementation-defined functions shall be recognized from the input data. It is
6999 implementation-defined how IEXTEN being set interacts with ICANON, ISIG, IXON, or IXOFF. |
7000 If IEXTEN is not set, implementation-defined functions shall not be recognized and the
7001 corresponding input characters are processed as described for ICANON, ISIG, IXON, and
7002 IXOFF.

7003 If ISIG is set, each input character shall be checked against the special control characters INTR,
7004 QUIT, and SUSP. If an input character matches one of these control characters, the function
7005 associated with that character shall be performed. If ISIG is not set, no checking shall be done.
7006 Thus these special input functions are possible only if ISIG is set.

7007 If NOFLSH is set, the normal flush of the input and output queues associated with the INTR,
7008 QUIT, and SUSP characters shall not be done.

7009 If TOSTOP is set, the signal SIGTTOU shall be sent to the process group of a process that tries to
7010 write to its controlling terminal if it is not in the foreground process group for that terminal. This
7011 signal, by default, stops the members of the process group. Otherwise, the output generated by |
7012 that process shall be output to the current output stream. Processes that are blocking or ignoring |
7013 SIGTTOU signals are excepted and allowed to produce output, and the SIGTTOU signal shall |
7014 not be sent. |

7015 The initial local control value after *open()* is implementation-defined.

7016 11.2.6 Special Control Characters

7017 The special control character values shall be defined by the array `c_cc`. The subscript name and |
7018 description for each element in both canonical and non-canonical modes are as follows:

7019
7020
7021
7022
7023
7024
7025
7026
7027
7028
7029
7030
7031
7032
7033

Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF		EOF character
VEOL		EOL character
VERASE		ERASE character
VINTR	VINTR	INTR character
VKILL		KILL character
	VMIN	MIN value
VQUIT	VQUIT	QUIT character
VSUSP	VSUSP	SUSP character
	VTIME	TIME value
VSTART	VSTART	START character
VSTOP	VSTOP	STOP character

7034
7035

The subscript values are unique, except that the VMIN and VTIME subscripts may have the same values as the VEOF and VEOL subscripts, respectively.

7036
7037
7038

Implementations that do not support changing the START and STOP characters may ignore the character values in the `c_cc` array indexed by the VSTART and VSTOP subscripts when `tcsetattr()` is called, but shall return the value in use when `tcgetattr()` is called.

7039

The initial values of all control characters are implementation-defined.

7040
7041
7042
7043

If the value of one of the changeable special control characters (see Section 11.1.9 (on page 187)) is `_POSIX_VDISABLE`, that function shall be disabled; that is, no input data is recognized as the disabled special character. If ICANON is not set, the value of `_POSIX_VDISABLE` has no special meaning for the VMIN and VTIME entries of the `c_cc` array.

7045

7046 **12.1 Utility Argument Syntax**

7047 This section describes the argument syntax of the standard utilities and introduces terminology
 7048 used throughout IEEE Std 1003.1-200x for describing the arguments processed by the utilities.

7049 Within IEEE Std 1003.1-200x, a special notation is used for describing the syntax of a utility's
 7050 arguments. Unless otherwise noted, all utility descriptions use this notation, which is illustrated
 7051 by this example (see the Shell and Utilities volume of IEEE Std 1003.1-200x, Section 2.9.1, Simple
 7052 Commands):

```
7053 utility_name[-a][-b][-c option_argument]
7054 [-d|-e][-foption_argument][operand...]
```

7055 The notation used for the SYNOPSIS sections imposes requirements on the implementors of the
 7056 standard utilities and provides a simple reference for the application developer or system user.

7057 1. The utility in the example is named *utility_name*. It is followed by *options*, *option-*
 7058 *arguments*, and *operands*. The arguments that consist of hyphens and single letters or
 7059 digits, such as 'a', are known as *options* (or, historically, *flags*). Certain options are
 7060 followed by an *option-argument*, as shown with [-c *option_argument*]. The arguments
 7061 following the last options and option-arguments are named *operands*.

7062 2. Option-arguments are sometimes shown separated from their options by <blank>s,
 7063 sometimes directly adjacent. This reflects the situation that in some cases an option-
 7064 argument is included within the same argument string as the option; in most cases it is the
 7065 next argument. The Utility Syntax Guidelines in Section 12.2 (on page 199) require that the
 7066 option be a separate argument from its option-argument, but there are some exceptions in
 7067 IEEE Std 1003.1-200x to ensure continued operation of historical applications:

7068 a. If the SYNOPSIS of a standard utility shows a space character between an option and
 7069 option-argument (as with [-c *option_argument*] in the example), a conforming
 7070 application shall use separate arguments for that option and its option-argument. |

7071 b. If a space character is not shown (as with [-*foption_argument*] in the example), a |
 7072 conforming application shall place an option and its option-argument directly |
 7073 adjacent in the same argument string, without intervening <blank>s.

7074 c. Notwithstanding the preceding requirements on conforming applications, a |
 7075 conforming system shall permit, but shall not require, an application to specify
 7076 options and option-arguments as separate arguments whether or not a space
 7077 XSI character is shown on the synopsis line, except in those cases (marked with the XSI
 7078 portability warning) where an option-argument is optional and no separation can be
 7079 used.

7080 d. A standard utility may also be implemented to operate correctly when the required
 7081 separation into multiple arguments is violated by a non-conforming application. |

7082 In summary, the following table shows allowable combinations:

7083
7084
7085
7086
7087
7088

	SYNOPSIS Shows:		
	-a <i>arg</i>	-b <i>arg</i>	-c[<i>arg</i>]
Conforming application shall use:	-a <i>arg</i>	-b <i>arg</i>	N/A
System shall support:	-a <i>arg</i>	-b <i>arg</i>	-c <i>arg</i> or -c
System may support:	-a <i>arg</i>	-b <i>arg</i>	

7089
7090
7091
7092
7093
7094
7095
7096
7097

- Options are usually listed in alphabetical order unless this would make the utility description more confusing. There are no implied relationships between the options based upon the order in which they appear, unless otherwise stated in the OPTIONS section, or unless the exception in Guideline 11 of Section 12.2 (on page 199) applies. If an option that does not have option-arguments is repeated, the results are undefined, unless otherwise stated.
- Frequently, names of parameters that require substitution by actual values are shown with embedded underscores. Alternatively, parameters are shown as follows:

<parameter name>

7098
7099
7100

The angle brackets are used for the symbolic grouping of a phrase representing a single parameter and conforming applications shall not include them in data submitted to the utility.

7101
7102
7103

- When a utility has only a few permissible options, they are sometimes shown individually, as in the example. Utilities with many flags generally show all of the individual flags (that do not take option-arguments) grouped, as in:

utility_name [-abcDxyz][*-p arg*][*operand*]

7104

Utilities with very complex arguments may be shown as follows:

7105

utility_name [*options*][*operands*]

7106
7107
7108

- Unless otherwise specified, whenever an operand or option-argument is, or contains, a numeric value:

7109

- The number is interpreted as a decimal integer.

7110

- Numerals in the range 0 to 2 147 483 647 are syntactically recognized as numeric values.

7111

- When the utility description states that it accepts negative numbers as operands or option-arguments, numerals in the range -2 147 483 647 to 2 147 483 647 are syntactically recognized as numeric values.

7112

7113

- Ranges greater than those listed here are allowed.

7114

This does not mean that all numbers within the allowable range are necessarily semantically correct. A standard utility that accepts an option-argument or operand that is to be interpreted as a number, and for which a range of values smaller than that shown above is permitted by the IEEE Std 1003.1-200x, describes that smaller range along with the description of the option-argument or operand. If an error is generated, the utility's diagnostic message shall indicate that the value is out of the supported range, not that it is syntactically incorrect.

7115

7116

7117

7118

- Arguments or option-arguments enclosed in the '[' and ']' notation are optional and can be omitted. Conforming applications shall not include the '[' and ']' symbols in data submitted to the utility.

7119

7120

7121

- Arguments separated by the '|' vertical bar notation are mutually-exclusive. Conforming applications shall not include the '|' symbol in data submitted to the utility.

7122

7123

7124

7127 Alternatively, mutually-exclusive options and operands may be listed with multiple
7128 synopsis lines. For example:

```
7129     utility_name -d[-a][-c option_argument][operand...]  
7130     utility_name[-a][-b][operand...]
```

7131 When multiple synopsis lines are given for a utility, it is an indication that the utility has
7132 mutually-exclusive arguments. These mutually-exclusive arguments alter the functionality
7133 of the utility so that only certain other arguments are valid in combination with one of the
7134 mutually-exclusive arguments. Only one of the mutually-exclusive arguments is allowed
7135 for invocation of the utility. Unless otherwise stated in an accompanying OPTIONS
7136 section, the relationships between arguments depicted in the SYNOPSIS sections are
7137 mandatory requirements placed on conforming applications. The use of conflicting
7138 mutually-exclusive arguments produces undefined results, unless a utility description
7139 specifies otherwise. When an option is shown without the '[' and ']' brackets, it means
7140 that option is required for that version of the SYNOPSIS. However, it is not required to be
7141 the first argument, as shown in the example above, unless otherwise stated.

7142 9. Ellipses ("...") are used to denote that one or more occurrences of an option or operand
7143 are allowed. When an option or an operand followed by ellipses is enclosed in brackets,
7144 zero or more options or operands can be specified. The forms:

```
7145     utility_name -f option_argument...[operand...]  
7146     utility_name [-g option_argument]...[operand...]
```

7147 indicate that multiple occurrences of the option and its option-argument preceding the
7148 ellipses are valid, with semantics as indicated in the OPTIONS section of the utility. (See
7149 also Guideline 11 in Section 12.2.) In the first example, each option-argument requires a
7150 preceding `-f` and at least one `-f option_argument` must be given.

7151 10. When the synopsis line is too long to be printed on a single line in the Shell and Utilities
7152 volume of IEEE Std 1003.1-200x, the indented lines following the initial line are
7153 continuation lines. An actual use of the command would appear on a single logical line.

7154 12.2 Utility Syntax Guidelines

7155 The following guidelines are established for the naming of utilities and for the specification of
7156 options, option-arguments, and operands. The `getopt()` function in the System Interfaces volume
7157 of IEEE Std 1003.1-200x assists utilities in handling options and operands that conform to these
7158 guidelines.

7159 Operands and option-arguments can contain characters not specified in the portable character
7160 set.

7161 The guidelines are intended to provide guidance to the authors of future utilities, such as those
7162 written specific to a local system or that are components of a larger application. Some of the
7163 standard utilities do not conform to all of these guidelines; in those cases, the OPTIONS sections
7164 describe the deviations.

7165 **Guideline 1:** Utility names should be between two and nine characters, inclusive.

7166 **Guideline 2:** Utility names should include lowercase letters (the **lower** character
7167 classification) and digits only from the portable character set.

7168 **Guideline 3:** Each option name should be a single alphanumeric character (the **alnum**
7169 character classification) from the portable character set. The `-W` (capital-W)
7170 option shall be reserved for vendor options.

- 7171 Multi-digit options should not be allowed. |
- 7172 **Guideline 4:** All options should be preceded by the '-' delimiter character.
- 7173 **Guideline 5:** Options without option-arguments should be accepted when grouped behind
7174 one '-' delimiter.
- 7175 **Guideline 6:** Each option and option-argument should be a separate argument, except as
7176 noted in Section 12.1 (on page 197), item (2).
- 7177 **Guideline 7:** Option-arguments should not be optional.
- 7178 **Guideline 8:** When multiple option-arguments are specified to follow a single option, they
7179 should be presented as a single argument, using commas within that
7180 argument or <blank>s within that argument to separate them.
- 7181 **Guideline 9:** All options should precede operands on the command line.
- 7182 **Guideline 10:** The argument -- should be accepted as a delimiter indicating the end of
7183 options. Any following arguments should be treated as operands, even if they
7184 begin with the '-' character. The -- argument should not be used as an
7185 option or as an operand.
- 7186 **Guideline 11:** The order of different options relative to one another should not matter,
7187 unless the options are documented as mutually-exclusive and such an option
7188 is documented to override any incompatible options preceding it. If an option
7189 that has option-arguments is repeated, the option and option-argument
7190 combinations should be interpreted in the order specified on the command
7191 line.
- 7192 **Guideline 12:** The order of operands may matter and position-related interpretations should
7193 be determined on a utility-specific basis.
- 7194 **Guideline 13:** For utilities that use operands to represent files to be opened for either reading
7195 or writing, the '-' operand should be used only to mean standard input (or
7196 standard output when it is clear from context that an output file is being
7197 specified).
- 7198 The utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x that claim conformance to
7199 these guidelines shall conform completely to these guidelines as if these guidelines contained the |
7200 term "shall" instead of "should". On some implementations, the utilities accept usage in |
7201 violation of these guidelines for backward compatibility as well as accepting the required form. |
- 7202 It is recommended that all future utilities and applications use these guidelines to enhance user
7203 portability. The fact that some historical utilities could not be changed (to avoid breaking |
7204 existing applications) should not deter this future goal. |

7205

7206 This chapter describes the contents of headers.

7207 Headers contain function prototypes, the definition of symbolic constants, common structures,
7208 preprocessor macros, and defined types. Each function in the System Interfaces volume of
7209 IEEE Std 1003.1-200x specifies the headers that an application shall include in order to use that
7210 function. In most cases, only one header is required. These headers are present on an application
7211 development system; they need not be present on the target execution system.

7212 **13.1 Format of Entries**

7213 The entries in this chapter are based on a common format as follows. The only sections relating
7214 to conformance are the SYNOPSIS and DESCRIPTION.

7215 **NAME**

7216 This section gives the name or names of the entry and briefly states its purpose.

7217 **SYNOPSIS**

7218 This section summarizes the use of the entry being described.

7219 **DESCRIPTION**

7220 This section describes the functionality of the header.

7221 **APPLICATION USAGE**

7222 This section is non-normative.

7223 This section gives warnings and advice to application writers about the entry. In the
7224 event of conflict between warnings and advice and a normative part of this volume of
7225 IEEE Std 1003.1-200x, the normative material is to be taken as correct.

7226 **RATIONALE**

7227 This section is non-normative.

7228 This section contains historical information concerning the contents of this volume of
7229 IEEE Std 1003.1-200x and why features were included or discarded by the standard
7230 developers.

7231 **FUTURE DIRECTIONS**

7232 This section is non-normative.

7233 This section provides comments which should be used as a guide to current thinking;
7234 there is not necessarily a commitment to adopt these future directions.

7235 **SEE ALSO**

7236 This section is non-normative.

7237 This section gives references to related information.

7238 **CHANGE HISTORY**

7239 This section is non-normative.

7240 This section shows the derivation of the entry and any significant changes that have
7241 been made to it.

7242 **NAME**7243 aio.h — asynchronous input and output (**REALTIME**)7244 **SYNOPSIS**

7245 AIO #include <aio.h>

7246

7247 **DESCRIPTION**7248 The <aio.h> header shall define the **aio** structure which shall include at least the following
7249 members:

7250	int	aio_fildes	File descriptor.
7251	off_t	aio_offset	File offset.
7252	volatile void	*aio_buf	Location of buffer.
7253	size_t	aio_nbytes	Length of transfer.
7254	int	aio_reqprio	Request priority offset.
7255	struct sigevent	aio_sigevent	Signal number and value.
7256	int	aio_lio_opcode	Operation to be performed.

7257 This header shall also include the following constants:

7258 **AIO_CANCELED** A return value indicating that all requested operations have been
7259 canceled.7260 **AIO_NOTCANCELED**7261 A return value indicating that some of the requested operations could not
7262 be canceled since they are in progress.7263 **AIO_ALLDONE** A return value indicating that none of the requested operations could be
7264 canceled since they are already complete.7265 **LIO_WAIT** A *lio_listio()* synchronization operation indicating that the calling thread
7266 is to suspend until the *lio_listio()* operation is complete.7267 **LIO_NOWAIT** A *lio_listio()* synchronization operation indicating that the calling thread
7268 is to continue execution while the *lio_listio()* operation is being
7269 performed, and no notification is given when the operation is complete.7270 **LIO_READ** A *lio_listio()* element operation option requesting a read.7271 **LIO_WRITE** A *lio_listio()* element operation option requesting a write.7272 **LIO_NOP** A *lio_listio()* element operation option indicating that no transfer is
7273 requested.7274 The following shall be declared as functions and may also be defined as macros. Function
7275 prototypes shall be provided. |

```

7276 int aio_cancel(int, struct aiocb *);
7277 int aio_error(const struct aiocb *);
7278 int aio_fsync(int, struct aiocb *);
7279 int aio_read(struct aiocb *);
7280 ssize_t aio_return(struct aiocb *);
7281 int aio_suspend(const struct aiocb *const[], int,
7282 const struct timespec *);
7283 int aio_write(struct aiocb *);
7284 int lio_listio(int, struct aiocb *restrict const[restrict], int,
7285 struct sigevent *restrict);

```

7286 Inclusion of the <aio.h> header may make visible symbols defined in the headers <fcntl.h>,
7287 <signal.h>, <sys/types.h>, and <time.h>.

7288 **APPLICATION USAGE**

7289 None.

7290 **RATIONALE**

7291 None.

7292 **FUTURE DIRECTIONS**

7293 None.

7294 **SEE ALSO**

7295 <fcntl.h>, <signal.h>, <sys/types.h>, <time.h>, the System Interfaces volume of
7296 IEEE Std 1003.1-200x, *fsync()*, *lseek()*, *read()*, *write()*

7297 **CHANGE HISTORY**

7298 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

7299 **Issue 6**

7300 The <aio.h> header is marked as part of the Asynchronous Input and Output option.

7301 The description of the constants is expanded.

7302 The **restrict** keyword is added to the prototype for *lio_listio()*.

7303 **NAME**

7304 arpa/inet.h — definitions for internet operations

7305 **SYNOPSIS**

7306 #include <arpa/inet.h>

7307 **DESCRIPTION**7308 The **in_port_t** and **in_addr_t** types shall be defined as described in <netinet/in.h>.7309 The **in_addr** structure shall be defined as described in <netinet/in.h>.7310 IP6 The **INET_ADDRSTRLEN** and **INET6_ADDRSTRLEN** macros shall be defined as described in |
7311 <netinet/in.h>.7312 The following shall either be declared as functions, defined as macros, or both. If functions are |
7313 declared, function prototypes shall be provided.

7314 uint32_t htonl(uint32_t);

7315 uint16_t htons(uint16_t);

7316 uint32_t ntohl(uint32_t);

7317 uint16_t ntohs(uint16_t);

7318 The **uint32_t** and **uint16_t** types shall be defined as described in <inttypes.h>.7319 The following shall be declared as functions and may also be defined as macros. Function |
7320 prototypes shall be provided.

7321 in_addr_t inet_addr(const char *);

7322 char *inet_ntoa(struct in_addr);

7323 const char *inet_ntop(int, const void *restrict, char *restrict,
7324 socklen_t);

7325 int inet_pton(int, const char *restrict, void *restrict);

7326 Inclusion of the <arpa/inet.h> header may also make visible all symbols from <netinet/in.h> |
7327 and <inttypes.h>.7328 **APPLICATION USAGE**

7329 None.

7330 **RATIONALE**

7331 None.

7332 **FUTURE DIRECTIONS**

7333 None.

7334 **SEE ALSO**7335 <netinet/in.h>, <inttypes.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *htonl()*,
7336 *inet_addr()*7337 **CHANGE HISTORY**

7338 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

7339 The **restrict** keyword is added to the prototypes for *inet_ntop()* and *inet_pton()*.

7340 **NAME**

7341 assert.h — verify program assertion

7342 **SYNOPSIS**

7343 #include <assert.h>

7344 **DESCRIPTION**

7345 cx The functionality described on this reference page is aligned with the ISO C standard. Any
7346 conflict between the requirements described here and the ISO C standard is unintentional. This
7347 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

7348 The <assert.h> header shall define the *assert()* macro. It refers to the macro NDEBUG which is
7349 not defined in the header. If NDEBUG is defined as a macro name before the inclusion of this
7350 header, the *assert()* macro shall be defined simply as:

7351 #define assert(ignore)((void) 0)

7352 Otherwise, the macro behaves as described in *assert()*.

7353 The *assert()* macro shall be redefined according to the current state of NDEBUG each time
7354 <assert.h> is included.

7355 The *assert()* macro shall be implemented as a macro, not as a function. If the macro definition is
7356 suppressed in order to access an actual function, the behavior is undefined.

7357 **APPLICATION USAGE**

7358 None.

7359 **RATIONALE**

7360 None.

7361 **FUTURE DIRECTIONS**

7362 None.

7363 **SEE ALSO**7364 The System Interfaces volume of IEEE Std 1003.1-200x, *assert()*7365 **CHANGE HISTORY**

7366 First released in Issue 1. Derived from Issue 1 of the SVID.

7367 **Issue 6**

7368 The definition of the *assert()* macro is changed for alignment with the ISO/IEC 9899:1999
7369 standard.

7370 **NAME**

7371 complex.h — complex arithmetic

7372 **SYNOPSIS**

7373 #include <complex.h>

7374 **DESCRIPTION**

7375 **cx** The functionality described on this reference page is aligned with the ISO C standard. Any
 7376 conflict between the requirements described here and the ISO C standard is unintentional. This
 7377 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

7378 The **<complex.h>** header shall define the following macros:7379 **complex** Expands to **_Complex**.7380 **_Complex_I** Expands to a constant expression of type **const float _Complex**, with the
7381 value of the imaginary unit (that is, a number such that $i^2=-1$).7382 **imaginary** Expands to **_Imaginary**.7383 **_Imaginary_I** Expands to a constant expression of type **const float _Imaginary** with the
7384 value of the imaginary unit.7385 **I** Expands to either **_Imaginary_I** or **_Complex_I**. If **_Imaginary_I** is not defined,
7386 **I** expands to **_Complex_I**.7387 The macros **imaginary** and **_Imaginary_I** shall be defined if and only if the implementation
7388 supports imaginary types.7389 An application may undefine and then, perhaps, redefine the **complex**, **imaginary**, and **I** macros.7390 The following shall be declared as functions and may also be defined as macros. Function
7391 prototypes shall be provided.

7392	double	cabs(double complex);
7393	float	cabsf(float complex);
7394	long double	cabsl(long double complex);
7395	double complex	acos(double complex);
7396	float complex	acosf(float complex);
7397	double complex	acosh(double complex);
7398	float complex	acoshf(float complex);
7399	long double complex	acoshl(long double complex);
7400	long double complex	acosl(long double complex);
7401	double	carg(double complex);
7402	float	cargf(float complex);
7403	long double	cargl(long double complex);
7404	double complex	casin(double complex);
7405	float complex	casinf(float complex);
7406	double complex	casinh(double complex);
7407	float complex	casinhf(float complex);
7408	long double complex	casinhl(long double complex);
7409	long double complex	casinl(long double complex);
7410	double complex	catan(double complex);
7411	float complex	catanf(float complex);
7412	double complex	catanh(double complex);
7413	float complex	catanhf(float complex);
7414	long double complex	catanhl(long double complex);
7415	long double complex	catanl(long double complex);

```

7416     double complex      ccos(double complex);
7417     float complex       ccosf(float complex);
7418     double complex      ccosh(double complex);
7419     float complex       ccoshf(float complex);
7420     long double complex ccoshl(long double complex);
7421     long double complex ccosl(long double complex);
7422     double complex      cexp(double complex);
7423     float complex       cexpf(float complex);
7424     long double complex cexpl(long double complex);
7425     double              cimag(double complex);
7426     float              cimagf(float complex);
7427     long double        cimagl(long double complex);
7428     double complex     clog(double complex);
7429     float complex      clogf(float complex);
7430     long double complex clogl(long double complex);
7431     double complex     conj(double complex);
7432     float complex      conjf(float complex);
7433     long double complex conjl(long double complex);
7434     double complex     cpow(double complex, double complex);
7435     float complex      cpowf(float complex, float complex);
7436     long double complex cpowl(long double complex, long double complex);
7437     double complex     cproj(double complex);
7438     float complex      cprojf(float complex);
7439     long double complex cprojl(long double complex);
7440     double             creal(double complex);
7441     float             crealf(float complex);
7442     long double        creall(long double complex);
7443     double complex     csin(double complex);
7444     float complex      csinf(float complex);
7445     double complex     csinh(double complex);
7446     float complex      csinhf(float complex);
7447     long double complex csinhl(long double complex);
7448     long double complex csinl(long double complex);
7449     double complex     csqrt(double complex);
7450     float complex      csqrtf(float complex);
7451     long double complex csqrtl(long double complex);
7452     double complex     ctan(double complex);
7453     float complex      ctanf(float complex);
7454     double complex     ctanh(double complex);
7455     float complex      ctanhf(float complex);
7456     long double complex ctanhl(long double complex);
7457     long double complex ctanl(long double complex);

```

7458 APPLICATION USAGE

7459 Values are interpreted as radians, not degrees.

7460 RATIONALE

7461 The choice of *I* instead of *i* for the imaginary unit concedes to the widespread use of the
7462 identifier *i* for other purposes. The application can use a different identifier, say *j*, for the
7463 imaginary unit by following the inclusion of the <complex.h> header with:

```

7464     #undef I
7465     #define j _Imaginary_I

```

7466 An *I* suffix to designate imaginary constants is not required, as multiplication by *I* provides a
7467 sufficiently convenient and more generally useful notation for imaginary terms. The
7468 corresponding real type for the imaginary unit is **float**, so that use of *I* for algorithmic or
7469 notational convenience will not result in widening types.

7470 On systems with imaginary types, the application has the ability to control whether use of the
7471 macro **I** introduces an imaginary type, by explicitly defining **I** to be `_Imaginary_I` or `_Complex_I`.
7472 Disallowing imaginary types is useful for some applications intended to run on implementations
7473 without support for such types.

7474 The macro `_Imaginary_I` provides a test for whether imaginary types are supported.

7475 The `cis()` function ($\cos(x) + I\sin(x)$) was considered but rejected because its implementation is
7476 easy and straightforward, even though some implementations could compute sine and cosine
7477 more efficiently in tandem.

7478 **FUTURE DIRECTIONS**

7479 The following function names and the same names suffixed with *f* or *l* are reserved for future
7480 use, and may be added to the declarations in the **<complex.h>** header.

7481	<code>cerf()</code>	<code>cexpm1()</code>	<code>clog2()</code>
7482	<code>cerfc()</code>	<code>clog10()</code>	<code>clgamma()</code>
7483	<code>cexp2()</code>	<code>clog1p()</code>	<code>ctgamma()</code>

7484 **SEE ALSO**

7485 The System Interfaces volume of IEEE Std 1003.1-200x, `cabs()`, `cacos()`, `cacosh()`, `carg()`, `casin()`,
7486 `casinh()`, `catan()`, `catanh()`, `ccos()`, `ccosh()`, `cexp()`, `cimag()`, `clog()`, `conj()`, `cpow()`, `cproj()`, `creal()`,
7487 `csin()`, `csinh()`, `csqrt()`, `ctan()`, `ctanh()`

7488 **CHANGE HISTORY**

7489 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

7490 **NAME**

7491 cpio.h — cpio archive values

7492 **SYNOPSIS**

7493 xSI #include <cpio.h>

7494

7495 **DESCRIPTION**

7496 Values needed by the *c_mode* field of the *cpio* archive format are described as follows:

7497

7498

Name	Description	Value (Octal)
C_IRUSR	Read by owner.	0000400
C_IWUSR	Write by owner.	0000200
C_IXUSR	Execute by owner.	0000100
C_IRGRP	Read by group.	0000040
C_IWGRP	Write by group.	0000020
C_IXGRP	Execute by group.	0000010
C_IROTH	Read by others.	0000004
C_IWOTH	Write by others.	0000002
C_IXOTH	Execute by others.	0000001
C_ISUID	Set user ID.	0004000
C_ISGID	Set group ID.	0002000
C_ISVTX	On directories, restricted deletion flag.	0001000
C_ISDIR	Directory.	0040000
C_ISFIFO	FIFO.	0010000
C_ISREG	Regular file.	0100000
C_ISBLK	Block special.	0060000
C_ISCHR	Character special.	0020000
C_ISCTG	Reserved.	0110000
C_ISLNK	Symbolic link.	0120000
C_ISSOCK	Socket.	0140000

7519 The header shall define the symbolic constant:

7520 MAGIC "070707"

7521 **APPLICATION USAGE**

7522 None.

7523 **RATIONALE**

7524 None.

7525 **FUTURE DIRECTIONS**

7526 None.

7527 **SEE ALSO**

7528 The Shell and Utilities volume of IEEE Std 1003.1-200x, *pax*

7529 **CHANGE HISTORY**

7530 First released in Issue 3 of the Headers Interface, Issue 3 specification. Derived from the
7531 POSIX.1-1988 standard.

7532 **Issue 6**

7533 The SEE ALSO is updated to refer to *pax*, since the *cpio* utility is not included in the Shell and
7534 Utilities volume of IEEE Std 1003.1-200x.

7535 **NAME**7536 `ctype.h` — character types7537 **SYNOPSIS**7538 `#include <ctype.h>`7539 **DESCRIPTION**

7540 **CX** Some of the functionality described on this reference page extends the ISO C standard.
7541 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
7542 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these
7543 symbols in this header.

7544 The following shall be declared as functions and may also be defined as macros. Function
7545 prototypes shall be provided.

7546 `int isalnum(int);`7547 `int isalpha(int);`7548 **XSI** `int isascii(int);`7549 `int isblank(int);`7550 `int iscntrl(int);`7551 `int isdigit(int);`7552 `int isgraph(int);`7553 `int islower(int);`7554 `int isprint(int);`7555 `int ispunct(int);`7556 `int isspace(int);`7557 `int isupper(int);`7558 `int isxdigit(int);`7559 **XSI** `int toascii(int);`7560 `int tolower(int);`7561 `int toupper(int);`

7562 The following are defined as macros:

7563 **XSI** `int _toupper(int);`7564 `int _tolower(int);`

7565

7566 **APPLICATION USAGE**

7567 None.

7568 **RATIONALE**

7569 None.

7570 **FUTURE DIRECTIONS**

7571 None.

7572 **SEE ALSO**

7573 **<locale.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *isalnum()*, *isalpha()*, *isascii()*,
7574 *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *mblen()*,
7575 *mbstowcs()*, *mbtowc()*, *setlocale()*, *toascii()*, *tolower()*, *_tolower()*, *toupper()*, *_toupper()*, *wcstombs()*,
7576 *wctomb()*

7577 **CHANGE HISTORY**

7578 First released in Issue 1. Derived from Issue 1 of the SVID.

7579 **Issue 6**

7580 Extensions beyond the ISO C standard are now marked.

7581 **NAME**7582 `dirent.h` — format of directory entries7583 **SYNOPSIS**7584 `#include <dirent.h>`7585 **DESCRIPTION**

7586 The internal format of directories is unspecified.

7587 The **<dirent.h>** header shall define the following type:7588 **DIR** A type representing a directory stream.7589 It shall also define the structure **dirent** which shall include the following members:7590 XSI `ino_t d_ino` File serial number.7591 `char d_name[]` Name of entry.7592 XSI The type `ino_t` shall be defined as described in **<sys/types.h>**.7593 The character array `d_name` is of unspecified size, but the number of bytes preceding the
7594 terminating null byte shall not exceed `{NAME_MAX}`.7595 The following shall be declared as functions and may also be defined as macros. Function
7596 prototypes shall be provided. |7597 `int closedir(DIR *);`7598 `DIR *opendir(const char *);`7599 `struct dirent *readdir(DIR *);`7600 TSF `int readdir_r(DIR *restrict, struct dirent *restrict,
7601 struct dirent **restrict);`7602 `void rewinddir(DIR *);`7603 XSI `void seekdir(DIR *, long);`7604 `long telldir(DIR *);`

7605

7606 **APPLICATION USAGE**

7607 None.

7608 **RATIONALE**7609 Information similar to that in the **<dirent.h>** header is contained in a file **<sys/dir.h>** in 4.2 BSD
7610 and 4.3 BSD. The equivalent in these implementations of **struct dirent** from this volume of
7611 IEEE Std 1003.1-200x is **struct direct**. The filename was changed because the name **<sys/dir.h>**
7612 was also used in earlier implementations to refer to definitions related to the older access
7613 method; this produced name conflicts. The name of the structure was changed because this
7614 volume of IEEE Std 1003.1-200x does not completely define what is in the structure, so it could
7615 be different on some implementations from **struct direct**.7616 The name of an array of **char** of an unspecified size should not be used as an lvalue. Use of: |7617 `sizeof(d_name)`

7618 is incorrect; use:

7619 `strlen(d_name)`

7620 instead.

7621 The array of **char** `d_name` is not a fixed size. Implementations may need to declare **struct dirent**
7622 with an array size for `d_name` of 1, but the actual number of characters provided matches (or
7623 only slightly exceeds) the length of the filename.

7624 **FUTURE DIRECTIONS**

7625 None.

7626 **SEE ALSO**7627 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *closedir()*, *opendir()*,
7628 *readdir()*, *readdir_r()*, *rewinddir()*, *seekdir()*, *telldir()*7629 **CHANGE HISTORY**

7630 First released in Issue 2.

7631 **Issue 5**

7632 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

7633 **Issue 6**7634 The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir_r()*.7635 The **restrict** keyword is added to the prototype for *readdir_r()*.

7636 **NAME**7637 `dlfcn.h` — dynamic linking7638 **SYNOPSIS**7639 XSI `#include <dlfcn.h>`

7640

7641 **DESCRIPTION**7642 The `<dlfcn.h>` header shall define at least the following macros for use in the construction of a
7643 `dlopen()` *mode* argument:7644 `RTLD_LAZY` Relocations are performed at an implementation-defined time.7645 `RTLD_NOW` Relocations are performed when the object is loaded.7646 `RTLD_GLOBAL` All symbols are available for relocation processing of other modules.7647 `RTLD_LOCAL` All symbols are not made available for relocation processing by other
7648 modules.7649 The following shall be declared as functions and may also be defined as macros. Function |
7650 prototypes shall be provided. |7651 `int dlclose(void *);`7652 `char *dlerror(void);`7653 `void *dlopen(const char *, int);`7654 `void *dlsym(void *restrict, const char *restrict);`7655 **APPLICATION USAGE**

7656 None.

7657 **RATIONALE**

7658 None.

7659 **FUTURE DIRECTIONS**

7660 None.

7661 **SEE ALSO**7662 The System Interfaces volume of IEEE Std 1003.1-200x, `dlopen()`, `dlclose()`, `dlsym()`, `dlerror()`7663 **CHANGE HISTORY**

7664 First released in Issue 5.

7665 **Issue 6**7666 The `restrict` keyword is added to the prototype for `dlsym()`.

7667 **NAME**

7668 errno.h — system error numbers

7669 **SYNOPSIS**

7670 #include <errno.h>

7671 **DESCRIPTION**

7672 **CX** Some of the functionality described on this reference page extends the ISO C standard. Any
 7673 conflict between the requirements described here and the ISO C standard is unintentional. This
 7674 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

7675 **CX** The ISO C standard only requires the symbols [EDOM], [EILSEQ], and [ERANGE] to be defined.

7676 The <errno.h> header shall provide a declaration for *errno* and give positive values for the
 7677 following symbolic constants. Their values shall be unique except as noted below:

7678	[E2BIG]	Argument list too long.
7679	[EACCES]	Permission denied.
7680	[EADDRINUSE]	Address in use.
7681	[EADDRNOTAVAIL]	Address not available.
7682	[EAFNOSUPPORT]	Address family not supported.
7683	[EAGAIN]	Resource unavailable, try again (may be the same value as
7684		[EWOULDBLOCK]).
7685	[EALREADY]	Connection already in progress.
7686	[EBADF]	Bad file descriptor.
7687	[EBADMSG]	Bad message.
7688	[EBUSY]	Device or resource busy.
7689	[ECANCELED]	Operation canceled.
7690	[ECHILD]	No child processes.
7691	[ECONNABORTED]	Connection aborted.
7692	[ECONNREFUSED]	Connection refused.
7693	[ECONNRESET]	Connection reset.
7694	[EDEADLK]	Resource deadlock would occur.
7695	[EDESTADDRREQ]	Destination address required.
7696	[EDOM]	Mathematics argument out of domain of function.
7697	[EDQUOT]	Reserved.
7698	[EEXIST]	File exists.
7699	[EFAULT]	Bad address.
7700	[EFBIG]	File too large.
7701	[EHOSTUNREACH]	Host is unreachable.
7702	[EIDRM]	Identifier removed.
7703	[EILSEQ]	Illegal byte sequence.

7704		[EINPROGRESS]	Operation in progress.
7705		[EINTR]	Interrupted function.
7706		[EINVAL]	Invalid argument.
7707		[EIO]	I/O error.
7708		[EISCONN]	Socket is connected.
7709		[EISDIR]	Is a directory.
7710		[ELOOP]	Too many levels of symbolic links.
7711		[EMFILE]	Too many open files.
7712		[EMLINK]	Too many links.
7713		[EMSGSIZE]	Message too large.
7714		[EMULTIHOP]	Reserved.
7715		[ENAMETOOLONG]	Filename too long.
7716		[ENETDOWN]	Network is down.
7717		[ENETUNREACH]	Network unreachable.
7718		[ENFILE]	Too many files open in system.
7719		[ENOBUFS]	No buffer space available.
7720	XSR	[ENODATA]	No message is available on the STREAM head read queue.
7721		[ENODEV]	No such device.
7722		[ENOENT]	No such file or directory.
7723		[ENOEXEC]	Executable file format error.
7724		[ENOLCK]	No locks available.
7725		[ENOLINK]	Reserved.
7726		[ENOMEM]	Not enough space.
7727		[ENOMSG]	No message of the desired type.
7728		[ENOPROTOPT]	Protocol not available.
7729		[ENOSPC]	No space left on device.
7730	XSR	[ENOSR]	No STREAM resources.
7731	XSR	[ENOSTR]	Not a STREAM.
7732		[ENOSYS]	Function not supported.
7733		[ENOTCONN]	The socket is not connected.
7734		[ENOTDIR]	Not a directory.
7735		[ENOTEMPTY]	Directory not empty.
7736		[ENOTSOCK]	Not a socket.
7737		[ENOTSUP]	Not supported.

7738	[ENOTTY]	Inappropriate I/O control operation.
7739	[ENXIO]	No such device or address.
7740	[EOPNOTSUPP]	Operation not supported on socket.
7741	[EOVERFLOW]	Value too large to be stored in data type.
7742	[EPERM]	Operation not permitted.
7743	[EPIPE]	Broken pipe.
7744	[EPROTO]	Protocol error.
7745	[EPROTONOSUPPORT]	
7746		Protocol not supported.
7747	[EPROTOTYPE]	Protocol wrong type for socket.
7748	[ERANGE]	Result too large.
7749	[EROFS]	Read-only file system.
7750	[ESPIPE]	Invalid seek.
7751	[ESRCH]	No such process.
7752	[ESTALE]	Reserved.
7753	XSR [ETIME]	Stream <i>ioctl()</i> timeout.
7754	[ETIMEDOUT]	Connection timed out.
7755	[ETXTBSY]	Text file busy.
7756	[EWOULDBLOCK]	Operation would block (may be the same value as [EAGAIN]).
7757	[EXDEV]	Cross-device link.
7758	APPLICATION USAGE	
7759	Additional error numbers may be defined on conforming systems; see the System Interfaces	
7760	volume of IEEE Std 1003.1-200x.	
7761	RATIONALE	
7762	None.	
7763	FUTURE DIRECTIONS	
7764	None.	
7765	SEE ALSO	
7766	The System Interfaces volume of IEEE Std 1003.1-200x, Section 2.3, Error Numbers	
7767	CHANGE HISTORY	
7768	First released in Issue 1. Derived from Issue 1 of the SVID.	
7769	Issue 5	
7770	Updated for alignment with the POSIX Realtime Extension.	
7771	Issue 6	
7772	The following new requirements on POSIX implementations derive from alignment with the	
7773	Single UNIX Specification:	
7774	<ul style="list-style-type: none"> • The majority of the error conditions previously marked as extensions are now mandatory, 	
7775	except for the STREAMS-related error conditions.	

7776
7777

Values for *errno* are now required to be distinct positive values rather than non-zero values. This change is for alignment with the ISO/IEC 9899:1999 standard.

7778 **NAME**

7779 fcntl.h — file control options

7780 **SYNOPSIS**

7781 #include <fcntl.h>

7782 **DESCRIPTION**

7783 The <fcntl.h> header shall define the following requests and arguments for use by the functions
7784 *fcntl()* and *open()*.

7785 Values for *cmd* used by *fcntl()* (the following values are unique) are as follows:

- 7786 F_DUPFD Duplicate file descriptor.
- 7787 F_GETFD Get file descriptor flags.
- 7788 F_SETFD Set file descriptor flags.
- 7789 F_GETFL Get file status flags and file access modes.
- 7790 F_SETFL Set file status flags.
- 7791 F_GETLK Get record locking information.
- 7792 F_SETLK Set record locking information.
- 7793 F_SETLKW Set record locking information; wait if blocked.
- 7794 F_GETOWN Get process or process group ID to receive SIGURG signals.
- 7795 F_SETOWN Set process or process group ID to receive SIGURG signals.

7796 File descriptor flags used for *fcntl()* are as follows:

- 7797 FD_CLOEXEC Close the file descriptor upon execution of an *exec* family function.

7798 Values for *l_type* used for record locking with *fcntl()* (the following values are unique) are as
7799 follows:

- 7800 F_RDLCK Shared or read lock.
- 7801 F_UNLCK Unlock.
- 7802 F_WRLCK Exclusive or write lock.

7803 XSI The values used for *l_whence*, *SEEK_SET*, *SEEK_CUR*, and *SEEK_END* shall be defined as
7804 described in <unistd.h>.

7805 The following values are file creation flags and are used in the *oflag* value to *open()*. They shall
7806 be bitwise-distinct. |

- 7807 O_CREAT Create file if it does not exist.
- 7808 O_EXCL Exclusive use flag.
- 7809 O_NOCTTY Do not assign controlling terminal.
- 7810 O_TRUNC Truncate flag.

7811 File status flags used for *open()* and *fcntl()* are as follows:

- 7812 O_APPEND Set append mode.
- 7813 SIO O_DSYNC Write according to synchronized I/O data integrity completion.
- 7814 O_NONBLOCK Non-blocking mode.

7815 SIO **O_RSYNC** Synchronized read I/O operations.

7816 **O_SYNC** Write according to synchronized I/O file integrity completion.

7817 Mask for use with file access modes is as follows:

7818 **O_ACCMODE** Mask for file access modes.

7819 File access modes used for *open()* and *fcntl()* are as follows:

7820 **O_RDONLY** Open for reading only.

7821 **O_RDWR** Open for reading and writing.

7822 **O_WRONLY** Open for writing only.

7823 XSI The symbolic names for file modes for use as values of **mode_t** shall be defined as described in
7824 <sys/stat.h>.

7825 ADV Values for *advice* used by *posix_fadvise()* are as follows:

7826 **POSIX_FADV_NORMAL**
7827 The application has no advice to give on its behavior with respect to the specified data. It is
7828 the default characteristic if no advice is given for an open file.

7829 **POSIX_FADV_SEQUENTIAL**
7830 The application expects to access the specified data sequentially from lower offsets to
7831 higher offsets.

7832 **POSIX_FADV_RANDOM**
7833 The application expects to access the specified data in a random order.

7834 **POSIX_FADV_WILLNEED**
7835 The application expects to access the specified data in the near future.

7836 **POSIX_FADV_DONTNEED**
7837 The application expects that it will not access the specified data in the near future.

7838 **POSIX_FADV_NOREUSE**
7839 The application expects to access the specified data once and then not reuse it thereafter.
7840

7841 The structure **flock** describes a file lock. It shall include the following members:

7842 short l_type Type of lock; F_RDLCK, F_WRLCK, F_UNLCK.

7843 short l_whence Flag for starting offset.

7844 off_t l_start Relative offset in bytes.

7845 off_t l_len Size; if 0 then until EOF.

7846 pid_t l_pid Process ID of the process holding the lock; returned with F_GETLK.

7847 The **mode_t**, **off_t**, and **pid_t** types shall be defined as described in <sys/types.h>.

7848 The following shall be declared as functions and may also be defined as macros. Function |
7849 prototypes shall be provided. |

7850 int creat(const char *, mode_t);

7851 int fcntl(int, int, ...);

7852 int open(const char *, int, ...);

7853 ADV int posix_fadvise(int, off_t, size_t, int);

7854 int posix_fallocate(int, off_t, size_t);

7855

7856 XSI Inclusion of the <fcntl.h> header may also make visible all symbols from <sys/stat.h> and
7857 <unistd.h>.

7858 **APPLICATION USAGE**

7859 None.

7860 **RATIONALE**

7861 None.

7862 **FUTURE DIRECTIONS**

7863 None.

7864 **SEE ALSO**

7865 <sys/stat.h>, <sys/types.h>, <unistd.h>, the System Interfaces volume of IEEE Std 1003.1-200x,
7866 *creat()*, *exec()*, *fcntl()*, *open()*, *posix_fadvise()*, *posix_fallocate()*, *posix_madvise()*

7867 **CHANGE HISTORY**

7868 First released in Issue 1. Derived from Issue 1 of the SVID.

7869 **Issue 5**

7870 The DESCRIPTION is updated for alignment with POSIX Realtime Extension.

7871 **Issue 6**

7872 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

7873 • O_DSYNC and O_RSYNC are marked as part of the Synchronized Input and Output option.

7874 The following new requirements on POSIX implementations derive from alignment with the
7875 Single UNIX Specification:

7876 • The definition of the **mode_t**, **off_t**, and **pid_t** types is mandated.

7877 The F_GETOWN and F_SETOWN values are added for sockets.

7878 The *posix_fadvise()*, *posix_fallocate()*, and *posix_madvise()* functions are added for alignment with
7879 IEEE Std 1003.1d-1999.

7880 IEEE PASC Interpretation 1003.1 #102 is applied moving the prototype for *posix_madvise()* to
7881 <sys/mman.h>.

7882 **NAME**7883 `fenv.h` — floating-point environment7884 **SYNOPSIS**7885 `#include <fenv.h>`7886 **DESCRIPTION**

7887 `cx` The functionality described on this reference page is aligned with the ISO C standard. Any
7888 conflict between the requirements described here and the ISO C standard is unintentional. This
7889 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

7890 The **<fenv.h>** header shall define the following data types through **typedef**:

7891 **fenv_t** Represents the entire floating-point environment. The floating-point environment
7892 refers collectively to any floating-point status flags and control modes supported
7893 by the implementation.

7894 **fexcept_t** Represents the floating-point status flags collectively, including any status the
7895 implementation associates with the flags. A floating-point status flag is a system
7896 variable whose value is set (but never cleared) when a floating-point exception is
7897 raised, which occurs as a side effect of exceptional floating-point arithmetic to
7898 provide auxiliary information. A floating-point control mode is a system variable
7899 whose value may be set by the user to affect the subsequent behavior of floating-
7900 point arithmetic.

7901 The **<fenv.h>** header shall define the following constants if and only if the implementation
7902 supports the floating-point exception by means of the floating-point functions *fwclearexcept()*,
7903 *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, and *fetestexcept()*. Each expands to an integer
7904 constant expression with values such that bitwise-inclusive ORs of all combinations of the
7905 constants result in distinct values.

7906 `FE_DIVBYZERO`
7907 `FE_INEXACT`
7908 `FE_INVALID`
7909 `FE_OVERFLOW`
7910 `FE_UNDERFLOW`

7911 The **<fenv.h>** header shall define the following constant, which is simply the bitwise-inclusive
7912 OR of all floating-point exception constants defined above:

7913 `FE_ALL_EXCEPT`

7914 The **<fenv.h>** header shall define the following constants if and only if the implementation
7915 supports getting and setting the represented rounding direction by means of the *fegetround()*
7916 and *fesetround()* functions. Each expands to an integer constant expression whose values are
7917 distinct non-negative values.

7918 `FE_DOWNWARD`
7919 `FE_TONEAREST`
7920 `FE_TOWARDZERO`
7921 `FE_UPWARD`

7922 The **<fenv.h>** header shall define the following constant, which represents the default floating-
7923 point environment (that is, the one installed at program startup) and has type pointer to const-
7924 qualified **fenv_t**. It can be used as an argument to the functions within the **<fenv.h>** header that
7925 manage the floating-point environment.

7926 `FE_DFL_ENV`

7927 The following shall be declared as functions and may also be defined as macros. Function |
 7928 prototypes shall be provided. |

```
7929 int feclearexcept(int);
7930 int fegetexceptflag(fexcept_t *, int);
7931 int feraiseexcept(int);
7932 int fesetexceptflag(const fexcept_t *, int);
7933 int fetestexcept(int);
7934 int fegetround(void);
7935 int fesetround(int);
7936 int fegetenv(fenv_t *);
7937 int feholdexcept(fenv_t *);
7938 int fesetenv(const fenv_t *);
7939 int feupdateenv(const fenv_t *);
```

7940 The FENV_ACCESS pragma provides a means to inform the implementation when an
 7941 application might access the floating-point environment to test floating-point status flags or run
 7942 under non-default floating-point control modes. The pragma shall occur either outside external
 7943 declarations or preceding all explicit declarations and statements inside a compound statement.
 7944 When outside external declarations, the pragma takes effect from its occurrence until another
 7945 FENV_ACCESS pragma is encountered, or until the end of the translation unit. When inside a
 7946 compound statement, the pragma takes effect from its occurrence until another FENV_ACCESS
 7947 pragma is encountered (including within a nested compound statement), or until the end of the
 7948 compound statement; at the end of a compound statement the state for the pragma is restored to
 7949 its condition just before the compound statement. If this pragma is used in any other context, the
 7950 behavior is undefined. If part of an application tests floating-point status flags, sets floating-
 7951 point control modes, or runs under non-default mode settings, but was translated with the state
 7952 for the FENV_ACCESS pragma off, the behavior is undefined. The default state (on or off) for
 7953 the pragma is implementation-defined. (When execution passes from a part of the application
 7954 translated with FENV_ACCESS off to a part translated with FENV_ACCESS on, the state of the
 7955 floating-point status flags is unspecified and the floating-point control modes have their default
 7956 settings.)

7957 APPLICATION USAGE

7958 This header is designed to support the floating-point exception status flags and directed-
 7959 rounding control modes required by the IEC 60559:1989 standard, and other similar floating-
 7960 point state information. Also it is designed to facilitate code portability among all systems.

7961 Certain application programming conventions support the intended model of use for the
 7962 floating-point environment:

- 7963 • A function call does not alter its caller's floating-point control modes, clear its caller's
 7964 floating-point status flags, nor depend on the state of its caller's floating-point status flags
 7965 unless the function is so documented.
- 7966 • A function call is assumed to require default floating-point control modes, unless its
 7967 documentation promises otherwise.
- 7968 • A function call is assumed to have the potential for raising floating-point exceptions, unless
 7969 its documentation promises otherwise.

7970 With these conventions, an application can safely assume default floating-point control modes
 7971 (or be unaware of them). The responsibilities associated with accessing the floating-point
 7972 environment fall on the application that does so explicitly.

7973 Even though the rounding direction macros may expand to constants corresponding to the
 7974 values of FLT_ROUNDS, they are not required to do so.

```

7975     For example:
7976     #include <fenv.h>
7977     void f(double x)
7978     {
7979         #pragma STDC FENV_ACCESS ON
7980         void g(double);
7981         void h(double);
7982         /* ... */
7983         g(x + 1);
7984         h(x + 1);
7985         /* ... */
7986     }

```

7987 If the function *g()* might depend on status flags set as a side effect of the first *x+1*, or if the
7988 second *x+1* might depend on control modes set as a side effect of the call to function *g()*, then
7989 the application shall contain an appropriately placed invocation as follows:

```

7990     #pragma STDC FENV_ACCESS ON

```

7991 **RATIONALE**

7992 **The *fexcept_t* Type**

7993 **fexcept_t** does not have to be an integer type. Its values must be obtained by a call to
7994 *fegetexceptflag()*, and cannot be created by logical operations from the exception macros. An
7995 implementation might simply implement **fexcept_t** as an **int** and use the representations
7996 reflected by the exception macros, but is not required to; other representations might contain
7997 extra information about the exceptions. **fexcept_t** might be a **struct** with a member for each
7998 exception (that might hold the address of the first or last floating-point instruction that caused
7999 that exception). The ISO/IEC 9899:1999 standard makes no claims about the internals of an
8000 **fexcept_t**, and so the user cannot inspect it.

8001 **Exception and Rounding Macros**

8002 Macros corresponding to unsupported modes and rounding directions are not defined by the
8003 implementation and must not be defined by the application. An application might use **#ifdef** to
8004 test for this.

8005 **FUTURE DIRECTIONS**

8006 None.

8007 **SEE ALSO**

8008 The System Interfaces volume of IEEE Std 1003.1-200x, *feclearexcept()*, *fegetenv()*, *fegetexceptflag()*,
8009 *fegetround()*, *fehldexcept()*, *feraiseexcept()*, *fesetenv()*, *fesetexceptflag()*, *fesetround()*, *fetestexcept()*,
8010 *feupdateenv()*

8011 **CHANGE HISTORY**

8012 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

8013 The return types for *feclearexcept()*, *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, *fegetenv()*,
8014 *fesetenv()*, and *feupdateenv()* are changed from **void** to **int** for alignment with the
8015 ISO/IEC 9899:1999 standard, Defect Report 202.

8016 **NAME**

8017 float.h — floating types

8018 **SYNOPSIS**

8019 #include <float.h>

8020 **DESCRIPTION**

8021 cx The functionality described on this reference page is aligned with the ISO C standard. Any
 8022 conflict between the requirements described here and the ISO C standard is unintentional. This
 8023 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

8024 The characteristics of floating types are defined in terms of a model that describes a
 8025 representation of floating-point numbers and values that provide information about an
 8026 implementation's floating-point arithmetic.

8027 The following parameters are used to define the model for each floating-point type:

8028 *s* Sign (± 1).

8029 *b* Base or radix of exponent representation (an integer > 1).

8030 *e* Exponent (an integer between a minimum e_{\min} and a maximum e_{\max}).

8031 *p* Precision (the number of base-*b* digits in the significand).

8032 f_k Non-negative integers less than *b* (the significand digits).

8033 A floating-point number *x* is defined by the following model:

8034
$$x = sb^e \sum_{k=1}^p f_k b^{-k}, e_{\min} \leq e \leq e_{\max}$$

8035 In addition to normalized floating-point numbers ($f_1 > 0$ if $x \neq 0$), floating types may be able to
 8036 contain other kinds of floating-point numbers, such as subnormal floating-point numbers ($x \neq 0$,
 8037 $e = e_{\min}$, $f_1 = 0$) and unnormalized floating-point numbers ($x \neq 0$, $e > e_{\min}$, $f_1 = 0$), and values that are
 8038 not floating-point numbers, such as infinities and NaNs. A NaN is an encoding signifying Not-
 8039 a-Number. A *quiet NaN* propagates through almost every arithmetic operation without raising a
 8040 floating-point exception; a *signaling NaN* generally raises a floating-point exception when
 8041 occurring as an arithmetic operand.

8042 The accuracy of the floating-point operations ('+', '-', '*', '/') and of the library functions
 8043 in <math.h> and <complex.h> that return floating-point results is implementation-defined. The
 8044 implementation may state that the accuracy is unknown.

8045 All integer values in the <float.h> header, except FLT_ROUNDS, shall be constant expressions
 8046 suitable for use in #if preprocessing directives; all floating values shall be constant expressions.
 8047 All except DECIMAL_DIG, FLT_EVAL_METHOD, FLT_RADIX, and FLT_ROUNDS have
 8048 separate names for all three floating-point types. The floating-point model representation is
 8049 provided for all values except FLT_EVAL_METHOD and FLT_ROUNDS.

8050 The rounding mode for floating-point addition is characterized by the implementation-defined
 8051 value of FLT_ROUNDS:

8052 -1 Indeterminable.

8053 0 Toward zero.

8054 1 To nearest.

8055 2 Toward positive infinity.

8056 3 Toward negative infinity.

8057 All other values for FLT_ROUNDS characterize implementation-defined rounding behavior.

8058 The values of operations with floating operands and values subject to the usual arithmetic
8059 conversions and of floating constants are evaluated to a format whose range and precision may
8060 be greater than required by the type. The use of evaluation formats is characterized by the
8061 implementation-defined value of FLT_EVAL_METHOD:

8062 -1 Indeterminable.

8063 0 Evaluate all operations and constants just to the range and precision of the type.

8064 1 Evaluate operations and constants of type **float** and **double** to the range and precision of the
8065 **double** type, evaluate **long double** operations and constants to the range and precision of
8066 the **long double** type.

8067 2 Evaluate all operations and constants to the range and precision of the **long double** type.

8068 All other negative values for FLT_EVAL_METHOD characterize implementation-defined
8069 behavior.

8070 The values given in the following list shall be defined as constant expressions with
8071 implementation-defined values that are greater or equal in magnitude (absolute value) to those
8072 shown, with the same sign.

8073 • Radix of exponent representation, *b*.

8074 FLT_RADIX 2

8075 • Number of base-FLT_RADIX digits in the floating-point significand, *p*.

8076 FLT_MANT_DIG

8077 DBL_MANT_DIG

8078 LDBL_MANT_DIG

8079 • Number of decimal digits, *n*, such that any floating-point number in the widest supported
8080 floating type with p_{\max} radix *b* digits can be rounded to a floating-point number with *n*
8081 decimal digits and back again without change to the value.

8082
$$\left\{ \begin{array}{ll} p_{\max} \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \left\lceil 1 + p_{\max} \log_{10} b \right\rceil & \text{otherwise} \end{array} \right.$$

8083 DECIMAL_DIG 10

8084 • Number of decimal digits, *q*, such that any floating-point number with *q* decimal digits can
8085 be rounded into a floating-point number with *p* radix *b* digits and back again without change
8086 to the *q* decimal digits.

8087
$$\left\{ \begin{array}{ll} p \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \left\lceil (p - 1) \log_{10} b \right\rceil & \text{otherwise} \end{array} \right.$$

8088 FLT_DIG 6

8089 DBL_DIG 10

8090 LDBL_DIG 10

8091 • Minimum negative integer such that FLT_RADIX raised to that power minus 1 is a
8092 normalized floating-point number, e_{\min} .

8093 FLT_MIN_EXP

8094 DBL_MIN_EXP

8095 LDBL_MIN_EXP

8096 • Minimum negative integer such that 10 raised to that power is in the range of normalized
8097 floating-point numbers.

8098 $\left\lceil \log_{10} b^{e_{\min} - 1} \right\rceil$

8099 FLT_MIN_10_EXP -37

8100 DBL_MIN_10_EXP -37

8101 LDBL_MIN_10_EXP -37

8102 • Maximum integer such that FLT_RADIX raised to that power minus 1 is a representable
8103 finite floating-point number, e_{\max} .

8104 FLT_MAX_EXP

8105 DBL_MAX_EXP

8106 LDBL_MAX_EXP

8107 • Maximum integer such that 10 raised to that power is in the range of representable finite
8108 floating-point numbers.

8109 $\left\lceil \log_{10} ((1 - b^{-p}) b^{e_{\max}}) \right\rceil$

8110 FLT_MAX_10_EXP +37

8111 DBL_MAX_10_EXP +37

8112 LDBL_MAX_10_EXP +37

8113 The values given in the following list shall be defined as constant expressions with
8114 implementation-defined values that are greater than or equal to those shown:

8115 • Maximum representable finite floating-point number.

8116 $(1 - b^{-p}) b^{e_{\max}}$

8117 FLT_MAX 1E+37

8118 DBL_MAX 1E+37

8119 LDBL_MAX 1E+37

8120 The values given in the following list shall be defined as constant expressions with
8121 implementation-defined (positive) values that are less than or equal to those shown:

8122 • The difference between 1 and the least value greater than 1 that is representable in the given
8123 floating-point type, b^{1-p} .

8124 FLT_EPSILON 1E-5

8125 DBL_EPSILON 1E-9

8126 LDBL_EPSILON 1E-9
8127 • Minimum normalized positive floating-point number, $b^{e_{\min} - 1}$.
8128 FLT_MIN 1E-37
8129 DBL_MIN 1E-37
8130 LDBL_MIN 1E-37

8131 **APPLICATION USAGE**
8132 None.

8133 **RATIONALE**
8134 None.

8135 **FUTURE DIRECTIONS**
8136 None.

8137 **SEE ALSO**
8138 <complex.h>, <math.h>

8139 **CHANGE HISTORY**
8140 First released in Issue 4. Derived from the ISO C standard.

8141 **Issue 6**
8142 The description of the operations with floating-point values is updated for alignment with the
8143 ISO/IEC 9899:1999 standard.

8144 **NAME**

8145 fmtmsg.h — message display structures

8146 **SYNOPSIS**

8147 xSI #include <fmtmsg.h>

8148

8149 **DESCRIPTION**

8150 The <fmtmsg.h> header shall define the following macros, which expand to constant integer
8151 expressions:

- 8152 MM_HARD Source of the condition is hardware.
- 8153 MM_SOFT Source of the condition is software.
- 8154 MM_FIRM Source of the condition is firmware.
- 8155 MM_APPL Condition detected by application.
- 8156 MM_UTIL Condition detected by utility.
- 8157 MM_OPSYS Condition detected by operating system.
- 8158 MM_RECOVER Recoverable error.
- 8159 MM_NRECOV Non-recoverable error.
- 8160 MM_HALT Error causing application to halt.
- 8161 MM_ERROR Application has encountered a non-fatal fault.
- 8162 MM_WARNING Application has detected unusual non-error condition.
- 8163 MM_INFO Informative message.
- 8164 MM_NOSEV No severity level provided for the message.
- 8165 MM_PRINT Display message on standard error.
- 8166 MM_CONSOLE Display message on system console.

8167 The table below indicates the null values and identifiers for *fmtmsg()* arguments. The
8168 <fmtmsg.h> header shall define the macros in the **Identifier** column, which expand to constant
8169 expressions that expand to expressions of the type indicated in the **Type** column:

8170

8171

Argument	Type	Null-Value	Identifier
<i>label</i>	char *	(char*)0	MM_NULLLBL
<i>severity</i>	int	0	MM_NULLSEV
<i>class</i>	long	0L	MM_NULLMC
<i>text</i>	char *	(char*)0	MM_NULLTXT
<i>action</i>	char *	(char*)0	MM_NULLACT
<i>tag</i>	char *	(char*)0	MM_NULLTAG

8172

8173

8174

8175

8176

8177

8178 The <fmtmsg.h> header shall also define the following macros for use as return values for
8179 *fmtmsg()*:

- 8180 MM_OK The function succeeded.
- 8181 MM_NOTOK The function failed completely.
- 8182 MM_NOMSG The function was unable to generate a message on standard error, but
8183 otherwise succeeded.

8200 **NAME**

8201 fnmatch.h — filename-matching types

8202 **SYNOPSIS**

8203 #include <fnmatch.h>

8204 **DESCRIPTION**

8205 The <fnmatch.h> header shall define the following constants:

8206 FNM_NOMATCH The string does not match the specified pattern.

8207 FNM_PATHNAME Slash in *string* only matches slash in *pattern*.8208 FNM_PERIOD Leading period in *string* must be exactly matched by period in *pattern*.

8209 FNM_NOESCAPE Disable backslash escaping.

8210 OB XSI FNM_NOSYS Reserved.

8211 The following shall be declared as a function and may also be defined as a macro. A function |
8212 prototype shall be provided. |

8213 int fnmatch(const char *, const char *, int);

8214 **APPLICATION USAGE**

8215 None.

8216 **RATIONALE**

8217 None.

8218 **FUTURE DIRECTIONS**

8219 None.

8220 **SEE ALSO**8221 The System Interfaces volume of IEEE Std 1003.1-200x, *fnmatch()*, the Shell and Utilities volume
8222 of IEEE Std 1003.1-200x8223 **CHANGE HISTORY**

8224 First released in Issue 4. Derived from the ISO POSIX-2 standard.

8225 **Issue 6**

8226 The constant FNM_NOSYS is marked obsolescent.

8227 **NAME**

8228 ftw.h — file tree traversal

8229 **SYNOPSIS**8230 XSI `#include <ftw.h>`

8231

8232 **DESCRIPTION**8233 The **<ftw.h>** header shall define the **FTW** structure that includes at least the following members:

8234 int base

8235 int level

8236 The **<ftw.h>** header shall define macros for use as values of the third argument to the
8237 application-supplied function that is passed as the second argument to *ftw()* and *nftw()*:8238 **FTW_F** File.8239 **FTW_D** Directory.8240 **FTW_DNR** Directory without read permission.8241 **FTW_DP** Directory with subdirectories visited.8242 **FTW_NS** Unknown type; *stat()* failed.8243 **FTW_SL** Symbolic link.8244 **FTW_SLN** Symbolic link that names a nonexistent file.8245 The **<ftw.h>** header shall define macros for use as values of the fourth argument to *nftw()*:8246 **FTW_PHYS** Physical walk, does not follow symbolic links. Otherwise, *nftw()* follows
8247 links but does not walk down any path that crosses itself.8248 **FTW_MOUNT** The walk does not cross a mount point.8249 **FTW_DEPTH** All subdirectories are visited before the directory itself.8250 **FTW_CHDIR** The walk changes to each directory before reading it.8251 The following shall be declared as functions and may also be defined as macros. Function |
8252 prototypes shall be provided. |8253 int ftw(const char *,
8254 int (*)(const char *, const struct stat *, int), int);
8255 int nftw(const char *, int (*)(
8256 const char *, const struct stat *, int, struct FTW*),
8257 int, int);8258 The **<ftw.h>** header shall define the **stat** structure and the symbolic names for *st_mode* and the
8259 file type test macros as described in **<sys/stat.h>**.8260 Inclusion of the **<ftw.h>** header may also make visible all symbols from **<sys/stat.h>**.

8261 **APPLICATION USAGE**

8262 None.

8263 **RATIONALE**

8264 None.

8265 **FUTURE DIRECTIONS**

8266 None.

8267 **SEE ALSO**8268 <sys/stat.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *ftw()*, *nftw()*8269 **CHANGE HISTORY**

8270 First released in Issue 1. Derived from Issue 1 of the SVID.

8271 **Issue 5**

8272 A description of FTW_DP is added.

8273 **NAME**

8274 glob.h — pathname pattern-matching types |

8275 **SYNOPSIS**

8276 #include <glob.h>

8277 **DESCRIPTION**8278 The <glob.h> header shall define the structures and symbolic constants used by the *glob()*
8279 function.8280 The structure type **glob_t** shall contain at least the following members:8281 size_t gl_pathc Count of paths matched by *pattern*.

8282 char **gl_pathv Pointer to a list of matched pathnames. |

8283 size_t gl_offs Slots to reserve at the beginning of *gl_pathv*. |8284 The following constants shall be provided as values for the *flags* argument:

8285 GLOB_APPEND Append generated pathnames to those previously obtained. |

8286 GLOB_DOOFFS Specify how many null pointers to add to the beginning of *pglob-*
8287 *>gl_pathv*.8288 GLOB_ERR Cause *glob()* to return on error.8289 GLOB_MARK Each pathname that is a directory that matches *pattern* has a slash |
8290 appended.8291 GLOB_NOCHECK If *pattern* does not match any pathname, then return a list consisting of |
8292 only *pattern*.

8293 GLOB_NOESCAPE Disable backslash escaping.

8294 GLOB_NOSORT Do not sort the pathnames returned. |

8295 The following constants shall be defined as error return values:

8296 GLOB_ABORTED The scan was stopped because GLOB_ERR was set or *(*errfunc)()*
8297 returned non-zero.8298 GLOB_NOMATCH The pattern does not match any existing pathname, and |
8299 GLOB_NOCHECK was not set in flags.

8300 GLOB_NOSPACE An attempt to allocate memory failed.

8301 OB XSI GLOB_NOSYS Reserved.

8302 The following shall be declared as functions and may also be defined as macros. Function |
8303 prototypes shall be provided.8304 int glob(const char *restrict, int, int (*restrict)(const char *, int),
8305 glob_t *restrict);

8306 void globfree (glob_t *);

8307 The implementation may define additional macros or constants using names beginning with
8308 GLOB_.

8309 **APPLICATION USAGE**

8310 None.

8311 **RATIONALE**

8312 None.

8313 **FUTURE DIRECTIONS**

8314 None.

8315 **SEE ALSO**8316 The System Interfaces volume of IEEE Std 1003.1-200x, *glob()*, the Shell and Utilities volume of
8317 IEEE Std 1003.1-200x8318 **CHANGE HISTORY**

8319 First released in Issue 4. Derived from the ISO POSIX-2 standard.

8320 **Issue 6**8321 The **restrict** keyword is added to the prototype for *glob()*.

8322 The constant GLOB_NOSYS is marked obsolescent.

8323 **NAME**

8324 grp.h — group structure

8325 **SYNOPSIS**

8326 #include <grp.h>

8327 **DESCRIPTION**

8328 The <grp.h> header shall declare the structure **group** which shall include the following
8329 members:

- 8330 char *gr_name The name of the group.
- 8331 gid_t gr_gid Numerical group ID.
- 8332 char **gr_mem Pointer to a null-terminated array of character
8333 pointers to member names.

8334 The **gid_t** type shall be defined as described in <sys/types.h>.

8335 The following shall be declared as functions and may also be defined as macros. Function |
8336 prototypes shall be provided. |

```

8337 struct group *getgrgid(gid_t);
8338 struct group *getgrnam(const char *);
8339 TSF int getgrgid_r(gid_t, struct group *, char *,
8340 size_t, struct group **);
8341 int getgrnam_r(const char *, struct group *, char *,
8342 size_t , struct group **);
8343 XSI struct group *getgrent(void);
8344 void endgrent(void);
8345 void setgrent(void);
8346

```

8347 **APPLICATION USAGE**

8348 None.

8349 **RATIONALE**

8350 None.

8351 **FUTURE DIRECTIONS**

8352 None.

8353 **SEE ALSO**

8354 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *endgrent()*, *getgrgid()*,
8355 *getgrnam()*

8356 **CHANGE HISTORY**

8357 First released in Issue 1.

8358 **Issue 5**

8359 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

8360 **Issue 6**

8361 The following new requirements on POSIX implementations derive from alignment with the
8362 Single UNIX Specification:

- 8363 • The definition of **gid_t** is mandated.
- 8364 • The *getgrgid_r()* and *getgrnam_r()* functions are marked as part of the Thread-Safe Functions
8365 option.

8366 **NAME**

8367 iconv.h — codeset conversion facility

8368 **SYNOPSIS**8369 XSI `#include <iconv.h>`

8370

8371 **DESCRIPTION**

8372 The <iconv.h> header shall define the following type:

8373 **iconv_t** Identifies the conversion from one codeset to another.8374 The following shall be declared as functions and may also be defined as macros. Function |
8375 prototypes shall be provided. |8376 `iconv_t iconv_open(const char *, const char *);`8377 `size_t iconv(iconv_t, char **restrict, size_t *restrict, char **restrict,`
8378 `size_t *restrict);`8379 `int iconv_close(iconv_t);`8380 **APPLICATION USAGE**

8381 None.

8382 **RATIONALE**

8383 None.

8384 **FUTURE DIRECTIONS**

8385 None.

8386 **SEE ALSO**8387 The System Interfaces volume of IEEE Std 1003.1-200x, *iconv()*, *iconv_close()*, *iconv_open()*8388 **CHANGE HISTORY**

8389 First released in Issue 4.

8390 **Issue 6**8391 The **restrict** keyword is added to the prototype for *iconv()*.

8392 NAME

8393 inttypes.h — fixed size integer types

8394 SYNOPSIS

8395 #include <inttypes.h>

8396 DESCRIPTION

8397 cx Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header.

8401 The <inttypes.h> header shall include the <stdint.h> header.

8402 The <inttypes.h> header shall include a definition of at least the following type:

8403 **imaxdiv_t** Structure type that is the type of the value returned by the *imaxdiv()* function.

8404 The following macros shall be defined. Each expands to a character string literal containing a conversion specifier, possibly modified by a length modifier, suitable for use within the *format* argument of a formatted input/output function when converting the corresponding integer type. These macros have the general form of PRI (character string literals for the *fprintf()* and *fwprintf()* family of functions) or SCN (character string literals for the *fscanf()* and *fwscanf()* family of functions), followed by the conversion specifier, followed by a name corresponding to a similar type name in <stdint.h>. In these names, *N* represents the width of the type as described in <stdint.h>. For example, *PRIdFAST32* can be used in a format string to print the value of an integer of type **int_fast32_t**.

8413 The *fprintf()* macros for signed integers are:

8414	PRIdN	PRIdLEASTN	PRIdFASTN	PRIdMAX	PRIdPTR
8415	PRiN	PRiLEASTN	PRiFASTN	PRiMAX	PRiPTR

8416 The *fprintf()* macros for unsigned integers are:

8417	PRIoN	PRIoLEASTN	PRIoFASTN	PRIoMAX	PRIoPTR
8418	PRiUN	PRiULEASTN	PRiUFASTN	PRiUMAX	PRiUPTR
8419	PRIxN	PRiXLEASTN	PRiXFASTN	PRiXMAX	PRiXPTR
8420	PRiXN	PRiXLEASTN	PRiXFASTN	PRiXMAX	PRiXPTR

8421 The *fscanf()* macros for signed integers are:

8422	SCNdN	SCNdLEASTN	SCNdFASTN	SCNdMAX	SCNdPTR
8423	SCNiN	SCNiLEASTN	SCNiFASTN	SCNiMAX	SCNiPTR

8424 The *fscanf()* macros for unsigned integers are:

8425	SCNoN	SCNoLEASTN	SCNoFASTN	SCNoMAX	SCNoPTR
8426	SCNuN	SCNuLEASTN	SCNuFASTN	SCNuMAX	SCNuPTR
8427	SCNxN	SCNxLEASTN	SCNxFASTN	SCNxMAX	SCNxPTR

8428 For each type that the implementation provides in <stdint.h>, the corresponding *fprintf()* macros shall be defined and the corresponding *fscanf()* macros shall be defined unless the implementation does not have a suitable *fscanf* length modifier for the type.

8431 The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided. |

```
8433 intmax_t imaxabs(intmax_t);
8434 imaxdiv_t imaxdiv(intmax_t, intmax_t);
8435 intmax_t strtoumax(const char *restrict, char **restrict, int);
```

```

8436     uintmax_t strtoumax(const char *restrict, char **restrict, int);
8437     intmax_t wcstoimax(const wchar_t *restrict, wchar_t **restrict, int);
8438     uintmax_t wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);

```

8439 EXAMPLES

```

8440     #include <inttypes.h>
8441     #include <wchar.h>
8442     int main(void)
8443     {
8444         uintmax_t i = UINTMAX_MAX; // This type always exists.
8445         wprintf(L"The largest integer value is %020"
8446             PRIxMAX "\n", i);
8447         return 0;
8448     }

```

8449 APPLICATION USAGE

8450 None.

8451 RATIONALE

8452 The ISO/IEC 9899:1990 standard specifies that the language should support four signed and
8453 unsigned integer data types—**char**, **short**, **int**, and **long**—but places very little requirement on
8454 their size other than that **int** and **short** be at least 16 bits and **long** be at least as long as **int** and
8455 not smaller than 32 bits. For 16-bit systems, most implementations assign 8, 16, 16, and 32 bits to
8456 **char**, **short**, **int**, and **long**, respectively. For 32-bit systems, the common practice is to assign 8, 16,
8457 32, and 32 bits to these types. This difference in **int** size can create some problems for users who
8458 migrate from one system to another which assigns different sizes to integer types, because the
8459 ISO C standard integer promotion rule can produce silent changes unexpectedly. The need for
8460 defining an extended integer type increased with the introduction of 64-bit systems.

8461 The purpose of <inttypes.h> is to provide a set of integer types whose definitions are consistent
8462 across machines and independent of operating systems and other implementation
8463 idiosyncrasies. It defines, via **typedef**, integer types of various sizes. Implementations are free to
8464 **typedef** them as ISO C standard integer types or extensions that they support. Consistent use of
8465 this header will greatly increase the portability of a users program across platforms.

8466 FUTURE DIRECTIONS

8467 Macro names beginning with PRI or SCN followed by any lowercase letter or 'x' may be added
8468 to the macros defined in the <inttypes.h> header.

8469 SEE ALSO

8470 The System Interfaces volume of IEEE Std 1003.1-200x, *imaxdiv()*

8471 CHANGE HISTORY

8472 First released in Issue 5.

8473 Issue 6

8474 The Open Group Base Resolution bwg97-006 is applied.

8475 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

8476 **NAME**

8477 iso646.h — alternative spellings

8478 **SYNOPSIS**

8479 #include <iso646.h>

8480 **DESCRIPTION**

8481 **cx** The functionality described on this reference page is aligned with the ISO C standard. Any
8482 conflict between the requirements described here and the ISO C standard is unintentional. This
8483 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

8484 The **<iso646.h>** header shall define the following eleven macros (on the left) that expand to the
8485 corresponding tokens (on the right):

8486 *and* &&8487 *and_eq* &=8488 *bitand* &8489 *bitor* |8490 *compl* ~8491 *not* !8492 *not_eq* !=8493 *or* | |8494 *or_eq* |=8495 *xor* ^8496 *xor_eq* ^=8497 **APPLICATION USAGE**

8498 None.

8499 **RATIONALE**

8500 None.

8501 **FUTURE DIRECTIONS**

8502 None.

8503 **SEE ALSO**

8504 None.

8505 **CHANGE HISTORY**

8506 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

8507 **NAME**

8508 langinfo.h — language information constants

8509 **SYNOPSIS**

8510 xSI #include <langinfo.h>

8511

8512 **DESCRIPTION**

8513 The <langinfo.h> header contains the constants used to identify items of *langinfo* data (see
8514 *nl_langinfo()*). The type of the constant, **nl_item**, shall be defined as described in <nl_types.h>.

8515 The following constants shall be defined. The entries under **Category** indicate in which
8516 *setlocale()* category each item is defined.

8517

Constant	Category	Meaning
8519 CODESET	<i>LC_CTYPE</i>	Codeset name.
8520 D_T_FMT	<i>LC_TIME</i>	String for formatting date and time.
8521 D_FMT	<i>LC_TIME</i>	Date format string.
8522 T_FMT	<i>LC_TIME</i>	Time format string.
8523 T_FMT_AMPM	<i>LC_TIME</i>	a.m. or p.m. time format string.
8524 AM_STR	<i>LC_TIME</i>	Ante Meridian affix.
8525 PM_STR	<i>LC_TIME</i>	Post Meridian affix.
8526 DAY_1	<i>LC_TIME</i>	Name of the first day of the week (for example, Sunday).
8527 DAY_2	<i>LC_TIME</i>	Name of the second day of the week (for example, Monday).
8528 DAY_3	<i>LC_TIME</i>	Name of the third day of the week (for example, Tuesday).
8529 DAY_4	<i>LC_TIME</i>	Name of the fourth day of the week (for example, Wednesday).
8530		
8531 DAY_5	<i>LC_TIME</i>	Name of the fifth day of the week (for example, Thursday).
8532 DAY_6	<i>LC_TIME</i>	Name of the sixth day of the week (for example, Friday).
8533 DAY_7	<i>LC_TIME</i>	Name of the seventh day of the week (for example, Saturday).
8534		
8535 ABDAY_1	<i>LC_TIME</i>	Abbreviated name of the first day of the week.
8536 ABDAY_2	<i>LC_TIME</i>	Abbreviated name of the second day of the week.
8537 ABDAY_3	<i>LC_TIME</i>	Abbreviated name of the third day of the week.
8538 ABDAY_4	<i>LC_TIME</i>	Abbreviated name of the fourth day of the week.
8539 ABDAY_5	<i>LC_TIME</i>	Abbreviated name of the fifth day of the week.
8540 ABDAY_6	<i>LC_TIME</i>	Abbreviated name of the sixth day of the week.
8541 ABDAY_7	<i>LC_TIME</i>	Abbreviated name of the seventh day of the week.
8542 MON_1	<i>LC_TIME</i>	Name of the first month of the year.
8543 MON_2	<i>LC_TIME</i>	Name of the second month.
8544 MON_3	<i>LC_TIME</i>	Name of the third month.
8545 MON_4	<i>LC_TIME</i>	Name of the fourth month.
8546 MON_5	<i>LC_TIME</i>	Name of the fifth month.
8547 MON_6	<i>LC_TIME</i>	Name of the sixth month.
8548 MON_7	<i>LC_TIME</i>	Name of the seventh month.
8549 MON_8	<i>LC_TIME</i>	Name of the eighth month.
8550 MON_9	<i>LC_TIME</i>	Name of the ninth month.
8551 MON_10	<i>LC_TIME</i>	Name of the tenth month.
8552 MON_11	<i>LC_TIME</i>	Name of the eleventh month.
8553 MON_12	<i>LC_TIME</i>	Name of the twelfth month.

8554
8555
8556
8557
8558
8559
8560
8561
8562
8563
8564
8565
8566
8567
8568
8569
8570
8571
8572
8573
8574
8575
8576
8577
8578
8579
8580

Constant	Category	Meaning
ABMON_1	LC_TIME	Abbreviated name of the first month.
ABMON_2	LC_TIME	Abbreviated name of the second month.
ABMON_3	LC_TIME	Abbreviated name of the third month.
ABMON_4	LC_TIME	Abbreviated name of the fourth month.
ABMON_5	LC_TIME	Abbreviated name of the fifth month.
ABMON_6	LC_TIME	Abbreviated name of the sixth month.
ABMON_7	LC_TIME	Abbreviated name of the seventh month.
ABMON_8	LC_TIME	Abbreviated name of the eighth month.
ABMON_9	LC_TIME	Abbreviated name of the ninth month.
ABMON_10	LC_TIME	Abbreviated name of the tenth month.
ABMON_11	LC_TIME	Abbreviated name of the eleventh month.
ABMON_12	LC_TIME	Abbreviated name of the twelfth month.
ERA	LC_TIME	Era description segments.
ERA_D_FMT	LC_TIME	Era date format string.
ERA_D_T_FMT	LC_TIME	Era date and time format string.
ERA_T_FMT	LC_TIME	Era time format string.
ALT_DIGITS	LC_TIME	Alternative symbols for digits.
RADIXCHAR	LC_NUMERIC	Radix character.
THOUSEP	LC_NUMERIC	Separator for thousands.
YESEXPR	LC_MESSAGES	Affirmative response expression.
NOEXPR	LC_MESSAGES	Negative response expression.
CRNCYSTR	LC_MONETARY	Currency symbol, preceded by '-' if the symbol should appear before the value, '+' if the symbol should appear after the value, or '.' if the symbol should replace the radix character.

8581 If the locale's values for **p_cs_precedes** and **n_cs_precedes** do not match, the value of
8582 *nl_langinfo*(CRNCYSTR) is unspecified.

8583 The following shall be declared as a function and may also be defined as a macro. A function
8584 prototype shall be provided.

```
8585 char *nl_langinfo(nl_item);
```

8586 Inclusion of the <langinfo.h> header may also make visible all symbols from <nl_types.h>.

8587 **APPLICATION USAGE**

8588 Wherever possible, users are advised to use functions compatible with those in the ISO C
8589 standard to access items of *langinfo* data. In particular, the *strptime*() function should be used to
8590 access date and time information defined in category *LC_TIME*. The *localeconv*() function
8591 should be used to access information corresponding to *RADIXCHAR*, *THOUSEP*, and
8592 *CRNCYSTR*.

8593 **RATIONALE**

8594 None.

8595 **FUTURE DIRECTIONS**

8596 None.

8597 **SEE ALSO**

8598 The System Interfaces volume of IEEE Std 1003.1-200x, *nl_langinfo*(), *localeconv*(), *strfmon*(),
8599 *strptime*(), Chapter 7 (on page 119)

8600 **CHANGE HISTORY**

8601 First released in Issue 2.

8602 **Issue 5**

8603 The constants YESSTR and NOSTR are marked LEGACY.

8604 **Issue 6**

8605 The constants YESSTR and NOSTR are removed.

8606 **NAME**

8607 libgen.h — definitions for pattern matching functions

8608 **SYNOPSIS**8609 XSI `#include <libgen.h>`

8610

8611 **DESCRIPTION**8612 The following shall be declared as functions and may also be defined as macros. Function |
8613 prototypes shall be provided. |8614 `char *basename(char *);`8615 `char *dirname(char *);`8616 **APPLICATION USAGE**

8617 None.

8618 **RATIONALE**

8619 None.

8620 **FUTURE DIRECTIONS**

8621 None.

8622 **SEE ALSO**8623 The System Interfaces volume of IEEE Std 1003.1-200x, *basename()*, *dirname()*8624 **CHANGE HISTORY**

8625 First released in Issue 4, Version 2.

8626 **Issue 5**8627 The function prototypes for *basename()* and *dirname()* are changed to indicate that the first
8628 argument is of type **char *** rather than **const char ***.8629 **Issue 6**8630 The `__loc1` symbol and the *regcmp()* and *regex()* functions are removed.

8631 **NAME**

8632 limits.h — implementation-defined constants

8633 **SYNOPSIS**

8634 #include <limits.h>

8635 **DESCRIPTION**

8636 CX Some of the functionality described on this reference page extends the ISO C standard.
 8637 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 8638 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these
 8639 symbols in this header.

8640 CX Many of the symbols listed here are not defined by the ISO/IEC 9899:1999 standard. Such
 8641 symbols are not shown as CX shaded.

8642 The <limits.h> header shall define various symbolic names. Different categories of names are
 8643 described below.

8644 The names represent various limits on resources that the implementation imposes on
 8645 applications.

8646 Implementations may choose any appropriate value for each limit, provided it is not more
 8647 restrictive than the Minimum Acceptable Values listed below. Symbolic constant names
 8648 beginning with _POSIX may be found in <unistd.h>.

8649 Applications should not assume any particular value for a limit. To achieve maximum
 8650 portability, an application should not require more resource than the Minimum Acceptable
 8651 Value quantity. However, an application wishing to avail itself of the full amount of a resource
 8652 available on an implementation may make use of the value given in <limits.h> on that
 8653 particular implementation, by using the symbolic names listed below. It should be noted,
 8654 however, that many of the listed limits are not invariant, and at runtime, the value of the limit
 8655 may differ from those given in this header, for the following reasons:

- 8656 • The limit is pathname-dependent.
- 8657 • The limit differs between the compile and runtime machines.

8658 For these reasons, an application may use the *fpathconf()*, *pathconf()*, and *sysconf()* functions to
 8659 determine the actual value of a limit at runtime.

8660 The items in the list ending in _MIN give the most negative values that the mathematical types
 8661 are guaranteed to be capable of representing. Numbers of a more negative value may be
 8662 supported on some implementations, as indicated by the <limits.h> header on the
 8663 implementation, but applications requiring such numbers are not guaranteed to be portable to
 8664 all implementations. For positive constants ending in _MIN, this indicates the minimum
 8665 acceptable value.

8666 The Minimum Acceptable Value symbol ' * ' indicates that there is no guaranteed value across
 8667 all conforming implementations.

8668 **Runtime Invariant Values (Possibly Indeterminate)**

8669 A definition of one of the symbolic names in the following list shall be omitted from **<limits.h>**
8670 on specific implementations where the corresponding value is equal to or greater than the stated
8671 minimum, but is unspecified.

8672 This indetermination might depend on the amount of available memory space on a specific
8673 instance of a specific implementation. The actual value supported by a specific instance shall be
8674 provided by the *sysconf()* function.

8675 AIO {AIO_LISTIO_MAX}
8676 Maximum number of I/O operations in a single list I/O call supported by the
8677 implementation.
8678 Minimum Acceptable Value: {_POSIX_AIO_LISTIO_MAX}

8679 AIO {AIO_MAX}
8680 Maximum number of outstanding asynchronous I/O operations supported by the
8681 implementation.
8682 Minimum Acceptable Value: {_POSIX_AIO_MAX}

8683 AIO {AIO_PRIO_DELTA_MAX}
8684 The maximum amount by which a process can decrease its asynchronous I/O priority level
8685 from its own scheduling priority.
8686 Minimum Acceptable Value: 0

8687 {ARG_MAX}
8688 Maximum length of argument to the *exec* functions including environment data.
8689 Minimum Acceptable Value: {_POSIX_ARG_MAX}

8690 XSI {ATEXIT_MAX}
8691 Maximum number of functions that may be registered with *atexit()*.
8692 Minimum Acceptable Value: 32

8693 {CHILD_MAX}
8694 Maximum number of simultaneous processes per real user ID.
8695 Minimum Acceptable Value: {_POSIX_CHILD_MAX}

8696 TMR {DELAYTIMER_MAX}
8697 Maximum number of timer expiration overruns.
8698 Minimum Acceptable Value: {_POSIX_DELAYTIMER_MAX}

8699 {HOST_NAME_MAX}
8700 Maximum length of a host name (not including the terminating null) as returned from the
8701 *gethostname()* function.
8702 Minimum Acceptable Value: {_POSIX_HOST_NAME_MAX}

8703 XSI {IOV_MAX}
8704 Maximum number of *iovec* structures that one process has available for use with *readv()* or
8705 *writev()*.
8706 Minimum Acceptable Value: {_XOPEN_IOV_MAX}

8707 {LOGIN_NAME_MAX}
8708 Maximum length of a login name.
8709 Minimum Acceptable Value: {_POSIX_LOGIN_NAME_MAX}

8710 MSG {MQ_OPEN_MAX}
8711 The maximum number of open message queue descriptors a process may hold.
8712 Minimum Acceptable Value: {_POSIX_MQ_OPEN_MAX}

8713	MSG	{MQ_PRIO_MAX}
8714		The maximum number of message priorities supported by the implementation.
8715		Minimum Acceptable Value: {_POSIX_MQ_PRIO_MAX}
8716		{OPEN_MAX}
8717		Maximum number of files that one process can have open at any one time.
8718		Minimum Acceptable Value: {_POSIX_OPEN_MAX}
8719		{PAGESIZE}
8720		Size in bytes of a page.
8721		Minimum Acceptable Value: 1
8722	XSI	{PAGE_SIZE}
8723		Equivalent to {PAGESIZE}. If either {PAGESIZE} or {PAGE_SIZE} is defined, the other is
8724		defined with the same value.
8725	THR	{PTHREAD_DESTRUCTOR_ITERATIONS}
8726		Maximum number of attempts made to destroy a thread's thread-specific data values on
8727		thread exit.
8728		Minimum Acceptable Value: {_POSIX_THREAD_DESTRUCTOR_ITERATIONS}
8729	THR	{PTHREAD_KEYS_MAX}
8730		Maximum number of data keys that can be created by a process.
8731		Minimum Acceptable Value: {_POSIX_THREAD_KEYS_MAX}
8732	THR	{PTHREAD_STACK_MIN}
8733		Minimum size in bytes of thread stack storage.
8734		Minimum Acceptable Value: 0
8735	THR	{PTHREAD_THREADS_MAX}
8736		Maximum number of threads that can be created per process.
8737		Minimum Acceptable Value: {_POSIX_THREAD_THREADS_MAX}
8738		{RE_DUP_MAX}
8739		The number of repeated occurrences of a BRE permitted by the <i>regex</i> (<i>c</i>) and <i>regcomp</i> (<i>c</i>)
8740		functions when using the interval notation <i>{\ (m,n)\}</i> ; see Section 9.3.6 (on page 170).
8741		Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}
8742	RTS	{RTSIG_MAX}
8743		Maximum number of realtime signals reserved for application use in this implementation.
8744		Minimum Acceptable Value: {_POSIX_RTSIG_MAX}
8745	SEM	{SEM_NSEMS_MAX}
8746		Maximum number of semaphores that a process may have.
8747		Minimum Acceptable Value: {_POSIX_SEM_NSEMS_MAX}
8748	SEM	{SEM_VALUE_MAX}
8749		The maximum value a semaphore may have.
8750		Minimum Acceptable Value: {_POSIX_SEM_VALUE_MAX}
8751	RTS	{SIGQUEUE_MAX}
8752		Maximum number of queued signals that a process may send and have pending at the
8753		receiver(s) at any time.
8754		Minimum Acceptable Value: {_POSIX_SIGQUEUE_MAX}
8755	SS TSP	{SS_REPL_MAX}
8756		The maximum number of replenishment operations that may be simultaneously pending
8757		for a particular sporadic server scheduler.
8758		Minimum Acceptable Value: {_POSIX_SS_REPL_MAX}

8759 {STREAM_MAX}
8760 The number of streams that one process can have open at one time. If defined, it has the
8761 same value as {FOPEN_MAX} (see <stdio.h>).
8762 Minimum Acceptable Value: {_POSIX_STREAM_MAX}

8763 {SYMLOOP_MAX}
8764 Maximum number of symbolic links that can be reliably traversed in the resolution of a
8765 pathname in the absence of a loop.
8766 Minimum Acceptable Value: {_POSIX_SYMLOOP_MAX}

8767 TMR {TIMER_MAX}
8768 Maximum number of timers per-process supported by the implementation.
8769 Minimum Acceptable Value: {_POSIX_TIMER_MAX}

8770 TRC {TRACE_EVENT_NAME_MAX}
8771 Maximum length of the trace event name.
8772 Minimum Acceptable Value: {_POSIX_TRACE_EVENT_NAME_MAX}

8773 TRC {TRACE_NAME_MAX}
8774 Maximum length of the trace generation version string or of the trace stream name.
8775 Minimum Acceptable Value: {_POSIX_TRACE_NAME_MAX}

8776 TRC {TRACE_SYS_MAX}
8777 Maximum number of trace streams that may simultaneously exist in the system.
8778 Minimum Acceptable Value: {_POSIX_TRACE_SYS_MAX}

8779 TRC {TRACE_USER_EVENT_MAX}
8780 Maximum number of user trace event type identifiers that may simultaneously exist in a
8781 traced process, including the predefined user trace event
8782 POSIX_TRACE_UNNAMED_USER_EVENT.
8783 Minimum Acceptable Value: {_POSIX_TRACE_USER_EVENT_MAX}

8784 {TTY_NAME_MAX}
8785 Maximum length of terminal device name.
8786 Minimum Acceptable Value: {_POSIX_TTY_NAME_MAX}

8787 {TZNAME_MAX}
8788 Maximum number of bytes supported for the name of a timezone (not of the TZ variable).
8789 Minimum Acceptable Value: {_POSIX_TZNAME_MAX}

8790 **Note:** The length given by {TZNAME_MAX} does not include the quoting characters mentioned in
8791 Section 8.3 (on page 161).

8792 Pathname Variable Values

8793 The values in the following list may be constants within an implementation or may vary from
8794 one pathname to another. For example, file systems or directories may have different
8795 characteristics.

8796 A definition of one of the values shall be omitted from the <limits.h> header on specific
8797 implementations where the corresponding value is equal to or greater than the stated minimum,
8798 but where the value can vary depending on the file to which it is applied. The actual value
8799 supported for a specific pathname shall be provided by the *pathconf()* function.

8800 {FILESIZEBITS}
8801 Minimum number of bits needed to represent, as a signed integer value, the maximum size
8802 of a regular file allowed in the specified directory.
8803 Minimum Acceptable Value: 32

8804 {LINK_MAX}
 8805 Maximum number of links to a single file.
 8806 Minimum Acceptable Value: {_POSIX_LINK_MAX}

8807 {MAX_CANON}
 8808 Maximum number of bytes in a terminal canonical input line.
 8809 Minimum Acceptable Value: {_POSIX_MAX_CANON}

8810 {MAX_INPUT}
 8811 Minimum number of bytes for which space is available in a terminal input queue; therefore, |
 8812 the maximum number of bytes a conforming application may require to be typed as input |
 8813 before reading them.
 8814 Minimum Acceptable Value: {_POSIX_MAX_INPUT}

8815 {NAME_MAX}
 8816 Maximum number of bytes in a filename (not including terminating null).
 8817 Minimum Acceptable Value: {_POSIX_NAME_MAX}
 8818 XSI Minimum Acceptable Value: {_XOPEN_NAME_MAX}

8819 {PATH_MAX}
 8820 Maximum number of bytes in a pathname, including the terminating null character. |
 8821 Minimum Acceptable Value: {_POSIX_PATH_MAX}
 8822 XSI Minimum Acceptable Value: {_XOPEN_PATH_MAX}

8823 {PIPE_BUF}
 8824 Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
 8825 Minimum Acceptable Value: {_POSIX_PIPE_BUF}

8826 ADV {POSIX_ALLOC_SIZE_MIN}
 8827 Minimum number of bytes of storage actually allocated for any portion of a file.
 8828 Minimum Acceptable Value: Not specified.

8829 ADV {POSIX_REC_INCR_XFER_SIZE}
 8830 Recommended increment for file transfer sizes between the
 8831 {POSIX_REC_MIN_XFER_SIZE} and {POSIX_REC_MAX_XFER_SIZE} values.
 8832 Minimum Acceptable Value: Not specified.

8833 ADV {POSIX_REC_MAX_XFER_SIZE}
 8834 Maximum recommended file transfer size.
 8835 Minimum Acceptable Value: Not specified.

8836 ADV {POSIX_REC_MIN_XFER_SIZE}
 8837 Minimum recommended file transfer size.
 8838 Minimum Acceptable Value: Not specified.

8839 ADV {POSIX_REC_XFER_ALIGN}
 8840 Recommended file transfer buffer alignment.
 8841 Minimum Acceptable Value: Not specified.

8842 {SYMLINK_MAX}
 8843 Maximum number of bytes in a symbolic link.
 8844 Minimum Acceptable Value: {_POSIX_SYMLINK_MAX}

8845 **Runtime Inceasable Values**

8846 The magnitude limitations in the following list shall be fixed by specific implementations. An
8847 application should assume that the value supplied by **<limits.h>** in a specific implementation is
8848 the minimum that pertains whenever the application is run under that implementation. A
8849 specific instance of a specific implementation may increase the value relative to that supplied by
8850 **<limits.h>** for that implementation. The actual value supported by a specific instance shall be
8851 provided by the *sysconf()* function.

8852 {BC_BASE_MAX}

8853 Maximum *obase* values allowed by the *bc* utility.
8854 Minimum Acceptable Value: {_POSIX2_BC_BASE_MAX}

8855 {BC_DIM_MAX}

8856 Maximum number of elements permitted in an array by the *bc* utility.
8857 Minimum Acceptable Value: {_POSIX2_BC_DIM_MAX}

8858 {BC_SCALE_MAX}

8859 Maximum *scale* value allowed by the *bc* utility.
8860 Minimum Acceptable Value: {_POSIX2_BC_SCALE_MAX}

8861 {BC_STRING_MAX}

8862 Maximum length of a string constant accepted by the *bc* utility.
8863 Minimum Acceptable Value: {_POSIX2_BC_STRING_MAX}

8864 {CHARCLASS_NAME_MAX}

8865 Maximum number of bytes in a character class name.
8866 Minimum Acceptable Value: {_POSIX2_CHARCLASS_NAME_MAX}

8867 {COLL_WEIGHTS_MAX}

8868 Maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order**
8869 keyword in the locale definition file; see Chapter 7 (on page 119).
8870 Minimum Acceptable Value: {_POSIX2_COLL_WEIGHTS_MAX}

8871 {EXPR_NEST_MAX}

8872 Maximum number of expressions that can be nested within parentheses by the *expr* utility.
8873 Minimum Acceptable Value: {_POSIX2_EXPR_NEST_MAX}

8874 {LINE_MAX}

8875 Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either
8876 standard input or another file), when the utility is described as processing text files. The
8877 length includes room for the trailing newline.
8878 Minimum Acceptable Value: {_POSIX2_LINE_MAX}

8879 {NGROUPS_MAX}

8880 Maximum number of simultaneous supplementary group IDs per process.
8881 Minimum Acceptable Value: {_POSIX2_NGROUPS_MAX}

8882 {RE_DUP_MAX}

8883 Maximum number of repeated occurrences of a regular expression permitted when using
8884 the interval notation $\{m,n\}$; see Chapter 9 (on page 165).
8885 Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}

8886 **Maximum Values**

8887 TMR The symbolic constants in the following list shall be defined in <limits.h> with the values
 8888 shown. These are symbolic names for the most restrictive value for certain features on an
 8889 implementation supporting the Timers option. A conforming implementation shall provide
 8890 values no larger than these values. A conforming application must not require a smaller value
 8891 for correct operation.

8892 TMR `{_POSIX_CLOCKRES_MIN}`
 8893 The resolution of the `CLOCK_REALTIME` clock, in nanoseconds.
 8894 Value: 20 000 000

8895 MON If the Monotonic Clock option is supported, the resolution of the `CLOCK_MONOTONIC`
 8896 clock, in nanoseconds, is represented by `{_POSIX_CLOCKRES_MIN}`.

8897 **Minimum Values**

8898 The symbolic constants in the following list shall be defined in <limits.h> with the values
 8899 shown. These are symbolic names for the most restrictive value for certain features on an
 8900 implementation conforming to this volume of IEEE Std 1003.1-200x. Related symbolic constants
 8901 are defined elsewhere in this volume of IEEE Std 1003.1-200x which reflect the actual
 8902 implementation and which need not be as restrictive. A conforming implementation shall
 8903 provide values at least this large. A strictly conforming application must not require a larger
 8904 value for correct operation.

8905 AIO `{_POSIX_AIO_LISTIO_MAX}`
 8906 The number of I/O operations that can be specified in a list I/O call.
 8907 Value: 2

8908 AIO `{_POSIX_AIO_MAX}`
 8909 The number of outstanding asynchronous I/O operations.
 8910 Value: 1

8911 `{_POSIX_ARG_MAX}`
 8912 Maximum length of argument to the `exec` functions including environment data.
 8913 Value: 4 096

8914 `{_POSIX_CHILD_MAX}`
 8915 Maximum number of simultaneous processes per real user ID.
 8916 Value: 6

8917 TMR `{_POSIX_DELAYTIMER_MAX}`
 8918 The number of timer expiration overruns.
 8919 Value: 32

8920 `{_POSIX_HOST_NAME_MAX}`
 8921 Maximum length of a host name (not including the terminating null) as returned from the
 8922 `gethostname()` function.
 8923 Value: 255

8924 `{_POSIX_LINK_MAX}`
 8925 Maximum number of links to a single file.
 8926 Value: 8

8927 `{_POSIX_LOGIN_NAME_MAX}`
 8928 The size of the storage required for a login name, in bytes, including the terminating null.
 8929 Value: 9

8930 { _POSIX_MAX_CANON}
8931 Maximum number of bytes in a terminal canonical input queue.
8932 Value: 255

8933 { _POSIX_MAX_INPUT}
8934 Maximum number of bytes allowed in a terminal input queue.
8935 Value: 255

8936 MSG { _POSIX_MQ_OPEN_MAX}
8937 The number of message queues that can be open for a single process.
8938 Value: 8

8939 MSG { _POSIX_MQ_PRIO_MAX}
8940 The maximum number of message priorities supported by the implementation.
8941 Value: 32

8942 { _POSIX_NAME_MAX}
8943 Maximum number of bytes in a filename (not including terminating null).
8944 Value: 14

8945 { _POSIX_NGROUPS_MAX}
8946 Maximum number of simultaneous supplementary group IDs per process.
8947 Value: 8

8948 { _POSIX_OPEN_MAX}
8949 Maximum number of files that one process can have open at any one time.
8950 Value: 20

8951 { _POSIX_PATH_MAX}
8952 Maximum number of bytes in a pathname. |
8953 Value: 256

8954 { _POSIX_PIPE_BUF}
8955 Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
8956 Value: 512

8957 { _POSIX_RE_DUP_MAX}
8958 The number of repeated occurrences of a BRE permitted by the *regex*(*)* and *regcomp*(*)*
8959 functions when using the interval notation *{\ (m,n)\}*; see Section 9.3.6 (on page 170).
8960 Value: 255

8961 RTS { _POSIX_RTSIG_MAX}
8962 The number of realtime signal numbers reserved for application use.
8963 Value: 8

8964 SEM { _POSIX_SEM_NSEMS_MAX}
8965 The number of semaphores that a process may have.
8966 Value: 256

8967 SEM { _POSIX_SEM_VALUE_MAX}
8968 The maximum value a semaphore may have.
8969 Value: 32 767

8970 RTS { _POSIX_SIGQUEUE_MAX}
8971 The number of queued signals that a process may send and have pending at the receiver(s)
8972 at any time.
8973 Value: 32

8974 { _POSIX_SSIZE_MAX }
 8975 The value that can be stored in an object of type `ssize_t`.
 8976 Value: 32 767

8977 { _POSIX_STREAM_MAX }
 8978 The number of streams that one process can have open at one time.
 8979 Value: 8

8980 SS|TSP { _POSIX_SS_REPL_MAX }
 8981 The number of replenishment operations that may be simultaneously pending for a
 8982 particular sporadic server scheduler.
 8983 Value: 4

8984 { _POSIX_SYMLINK_MAX }
 8985 The number of bytes in a symbolic link.
 8986 Value: 255

8987 { _POSIX_SYMLOOP_MAX }
 8988 The number of symbolic links that can be traversed in the resolution of a pathname in the
 8989 absence of a loop.
 8990 Value: 8

8991 THR { _POSIX_THREAD_DESTRUCTOR_ITERATIONS }
 8992 The number of attempts made to destroy a thread's thread-specific data values on thread
 8993 exit.
 8994 Value: 4

8995 THR { _POSIX_THREAD_KEYS_MAX }
 8996 The number of data keys per process.
 8997 Value: 128

8998 THR { _POSIX_THREAD_THREADS_MAX }
 8999 The number of threads per process.
 9000 Value: 64

9001 TMR { _POSIX_TIMER_MAX }
 9002 The per process number of timers.
 9003 Value: 32

9004 TRC { _POSIX_TRACE_EVENT_NAME_MAX }
 9005 The length in bytes of a trace event name.
 9006 Value: 30

9007 TRC { _POSIX_TRACE_NAME_MAX }
 9008 The length in bytes of a trace generation version string or a trace stream name.
 9009 Value: 8

9010 TRC { _POSIX_TRACE_SYS_MAX }
 9011 The number of trace streams that may simultaneously exist in the system.
 9012 Value: 8

9013 TRC { _POSIX_TRACE_USER_EVENT_MAX }
 9014 The number of user trace event type identifiers that may simultaneously exist in a traced
 9015 process, including the predefined user trace event
 9016 POSIX_TRACE_UNNAMED_USER_EVENT.
 9017 Value: 32

9018 { _POSIX_TTY_NAME_MAX }
 9019 The size of the storage required for a terminal device name, in bytes, including the

9020 terminating null.
9021 Value: 9

9022 `{_POSIX_TZNAME_MAX}`
9023 Maximum number of bytes supported for the name of a timezone (not of the TZ variable).
9024 Value: 6

9025 **Note:** The length given by `{_POSIX_TZNAME_MAX}` does not include the quoting characters
9026 mentioned in Section 8.3 (on page 161).

9027 `{_POSIX2_BC_BASE_MAX}`
9028 Maximum *obase* values allowed by the *bc* utility.
9029 Value: 99

9030 `{_POSIX2_BC_DIM_MAX}`
9031 Maximum number of elements permitted in an array by the *bc* utility.
9032 Value: 2 048

9033 `{_POSIX2_BC_SCALE_MAX}`
9034 Maximum *scale* value allowed by the *bc* utility.
9035 Value: 99

9036 `{_POSIX2_BC_STRING_MAX}`
9037 Maximum length of a string constant accepted by the *bc* utility.
9038 Value: 1 000

9039 `{_POSIX2_CHARCLASS_NAME_MAX}`
9040 Maximum number of bytes in a character class name.
9041 Value: 14

9042 `{_POSIX2_COLL_WEIGHTS_MAX}`
9043 Maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order**
9044 keyword in the locale definition file; see Chapter 7 (on page 119).
9045 Value: 2

9046 `{_POSIX2_EXPR_NEST_MAX}`
9047 Maximum number of expressions that can be nested within parentheses by the *expr* utility.
9048 Value: 32

9049 `{_POSIX2_LINE_MAX}`
9050 Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either
9051 standard input or another file), when the utility is described as processing text files. The
9052 length includes room for the trailing newline.
9053 Value: 2 048

9054 `{_POSIX2_RE_DUP_MAX}`
9055 Maximum number of repeated occurrences of a regular expression permitted when using
9056 the interval notation `\{m,n\}`; see Chapter 9 (on page 165).
9057 Value: 255

9058 XSI `{_XOPEN_IOV_MAX}`
9059 Maximum number of **iovec** structures that one process has available for use with *readv()* or
9060 *writev()*.
9061 Value: 16

9062 XSI `{_XOPEN_NAME_MAX}`
9063 Maximum number of bytes in a filename (not including terminating null).
9064 Value: 255

9065 XSI `{_XOPEN_PATH_MAX}`
 9066 Maximum number of bytes in a pathname.
 9067 Value: 1 024

9068 **Numerical Limits**

9069 The values in the following lists shall be defined in <limits.h> and are constant expressions
 9070 XSI suitable for use in #if preprocessing directives. Moreover, except for {CHAR_BIT}, {DBL_DIG},
 9071 {DBL_MAX}, {FLT_DIG}, {FLT_MAX}, {LONG_BIT}, {WORD_BIT}, and {MB_LEN_MAX}, the
 9072 symbolic names are defined as expressions of the correct type.

9073 If the value of an object of type **char** is treated as a signed integer when used in an expression,
 9074 the value of {CHAR_MIN} is the same as that of {SCHAR_MIN} and the value of {CHAR_MAX}
 9075 is the same as that of {SCHAR_MAX}. Otherwise, the value of {CHAR_MIN} is 0 and the value
 9076 of {CHAR_MAX} is the same as that of {UCHAR_MAX}.

9077 {CHAR_BIT}
 9078 Number of bits in a type **char**.
 9079 CX Value: 8

9080 {CHAR_MAX}
 9081 Maximum value of type **char**.
 9082 Minimum Acceptable Value: {UCHAR_MAX} or {SCHAR_MAX}

9083 {INT_MAX}
 9084 Maximum value of an **int**.
 9085 Minimum Acceptable Value: 2 147 483 647

9086 XSI `{LONG_BIT}`
 9087 Number of bits in a **long**.
 9088 Minimum Acceptable Value: 32

9089 {LONG_MAX}
 9090 Maximum value of a **long**.
 9091 Minimum Acceptable Value: +2 147 483 647

9092 {MB_LEN_MAX}
 9093 Maximum number of bytes in a character, for any supported locale.
 9094 Minimum Acceptable Value: 1

9095 {SCHAR_MAX}
 9096 Maximum value of type **signed char**.
 9097 CX Value: +127

9098 {SHRT_MAX}
 9099 Maximum value of type **short**.
 9100 Minimum Acceptable Value: +32 767

9101 {SSIZE_MAX}
 9102 Maximum value of an object of type **ssize_t**.
 9103 Minimum Acceptable Value: {_POSIX_SSIZE_MAX}

9104 {UCHAR_MAX}
 9105 Maximum value of type **unsigned char**.
 9106 CX Value: 255

9107 {UINT_MAX}
 9108 Maximum value of type **unsigned**.
 9109 Minimum Acceptable Value: 4 294 967 295

9110 {ULONG_MAX}
9111 Maximum value of type **unsigned long**.
9112 Minimum Acceptable Value: 4 294 967 295

9113 {USHRT_MAX}
9114 Maximum value for a type **unsigned short**.
9115 Minimum Acceptable Value: 65 535

9116 XSI {WORD_BIT}
9117 Number of bits in a word or type **int**.
9118 Minimum Acceptable Value: 16

9119 {CHAR_MIN}
9120 Minimum value of type **char**.
9121 Maximum Acceptable Value: {SCHAR_MIN} or 0

9122 {INT_MIN}
9123 Minimum value of type **int**.
9124 Maximum Acceptable Value: -2 147 483 647

9125 {LONG_MIN}
9126 Minimum value of type **long**.
9127 Maximum Acceptable Value: -2 147 483 647

9128 {SCHAR_MIN}
9129 Minimum value of type **signed char**.
9130 CX Value: -128

9131 {SHRT_MIN}
9132 Minimum value of type **short**.
9133 Maximum Acceptable Value: -32 767

9134 {LLONG_MIN}
9135 Minimum value of type **long long**.
9136 Maximum Acceptable Value: -9223372036854775807

9137 {LLONG_MAX}
9138 Maximum value of type **long long**.
9139 Minimum Acceptable Value: +9223372036854775807

9140 {ULLONG_MAX}
9141 Maximum value of type **unsigned long long**.
9142 Minimum Acceptable Value: 18446744073709551615

9143 Other Invariant Values

9144 XSI The following constants shall be defined on all implementations in **<limits.h>**:

9145 XSI {CHARCLASS_NAME_MAX}
9146 Maximum number of bytes in a character class name.
9147 Minimum Acceptable Value: 14

9148 XSI {NL_ARGMAX}
9149 Maximum value of *digit* in calls to the *printf()* and *scanf()* functions.
9150 Minimum Acceptable Value: 9

9151 XSI {NL_LANGMAX}
9152 Maximum number of bytes in a *LANG* name.
9153 Minimum Acceptable Value: 14

9154 XSI {NL_MSGMAX}
 9155 Maximum message number.
 9156 Minimum Acceptable Value: 32 767

9157 XSI {NL_NMAX}
 9158 Maximum number of bytes in an N-to-1 collation mapping.
 9159 Minimum Acceptable Value: ' * '

9160 XSI {NL_SETMAX}
 9161 Maximum set number.
 9162 Minimum Acceptable Value: 255

9163 XSI {NL_TEXTMAX}
 9164 Maximum number of bytes in a message string.
 9165 Minimum Acceptable Value: {_POSIX2_LINE_MAX}

9166 XSI {NZERO}
 9167 Default process priority.
 9168 Minimum Acceptable Value: 20

9169 **APPLICATION USAGE**

9170 None.

9171 **RATIONALE**

9172 A request was made to reduce the value of {_POSIX_LINK_MAX} from the value of 8 specified
 9173 for it in the POSIX.1-1990 standard to 2. The standard developers decided to deny this request
 9174 for several reasons.

9175 • They wanted to avoid making any changes to the standard that could break conforming
 9176 applications, and the requested change could have that effect.

9177 • The use of multiple hard links to a file cannot always be replaced with use of symbolic links.
 9178 Symbolic links are semantically different from hard links in that they associate a pathname
 9179 with another pathname rather than a pathname with a file. This has implications for access
 9180 control, file permanence, and transparency.

9181 • The original standard developers had considered the issue of allowing for implementations
 9182 that did not in general support hard links, and decided that this would reduce consensus on
 9183 the standard.

9184 Systems that support historical versions of the development option of the ISO POSIX-2 standard
 9185 retain the name {_POSIX2_RE_DUP_MAX} as an alias for {_POSIX_RE_DUP_MAX}.

9186 {PATH_MAX}
 9187 IEEE PASC Interpretation 1003.1 #15 addressed the inconsistency in the standard with the
 9188 definition of pathname and the description of {PATH_MAX}, allowing application writers to
 9189 allocate either {PATH_MAX} or {PATH_MAX}+1 bytes. The inconsistency has been
 9190 removed by correction to the {PATH_MAX} definition to include the null character. With
 9191 this change, applications that previously allocated {PATH_MAX} bytes will continue to
 9192 succeed.

9193 {SYMLINK_MAX}
 9194 This symbol refers to space for data that is stored in the file system, as opposed to
 9195 {PATH_MAX} which is the length of a name that can be passed to a function. In some
 9196 existing implementations, the filenames pointed to by symbolic links are stored in the
 9197 inodes of the links, so it is important that {SYMLINK_MAX} not be constrained to be as
 9198 large as {PATH_MAX}.

9199 FUTURE DIRECTIONS

9200 None.

9201 SEE ALSO

9202 The System Interfaces volume of IEEE Std 1003.1-200x, *fpathconf()*, *pathconf()*, *sysconf()*

9203 CHANGE HISTORY

9204 First released in Issue 1.

9205 Issue 5

9206 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
9207 Threads Extension.

9208 {FILESIZEBITS} added for the Large File Summit extensions.

9209 The minimum acceptable values for {INT_MAX}, {INT_MIN}, and {UINT_MAX} are changed to
9210 make 32-bit values the minimum requirement.

9211 The entry is restructured to improve readability.

9212 Issue 6

9213 The Open Group Corrigendum U033/4 is applied. The wording is made clear for {CHAR_MIN},
9214 {INT_MIN}, {LONG_MIN}, {SCHAR_MIN}, and {SHRT_MIN} that these are maximum
9215 acceptable values.

9216 The following new requirements on POSIX implementations derive from alignment with the
9217 Single UNIX Specification:

- 9218 • The minimum value for {CHILD_MAX} is 25. This is a FIPS requirement.
- 9219 • The minimum value for {OPEN_MAX} is 20. This is a FIPS requirement.
- 9220 • The minimum value for {NGROUPS_MAX} is 8. This is also a FIPS requirement.

9221 Symbolic constants are added for {_POSIX_SYMLINK_MAX}, {_POSIX_SYMLOOP_MAX},
9222 {_POSIX_RE_DUP_MAX}, {RE_DUP_MAX}, {SYMLOOP_MAX}, and {SYMLINK_MAX}.

9223 The following values are added for alignment with IEEE Std 1003.1d-1999:

9224 {_POSIX_SS_REPL_MAX}
9225 {SS_REPL_MAX}
9226 {POSIX_ALLOC_SIZE_MIN}
9227 {POSIX_REC_INCR_XFER_SIZE}
9228 {POSIX_REC_MAX_XFER_SIZE}
9229 {POSIX_REC_MIN_XFER_SIZE}
9230 {POSIX_REC_XFER_ALIGN}

9231 Reference to CLOCK_MONOTONIC is added in the description of {_POSIX_CLOCKRES_MIN}
9232 for alignment with IEEE Std 1003.1j-2000.

9233 The constants {LLONG_MIN}, {LLONG_MAX}, and {ULLONG_MAX} are added for alignment
9234 with the ISO/IEC 9899:1999 standard.

9235 The following values are added for alignment with IEEE Std 1003.1q-2000: |

9236	{_POSIX_TRACE_EVENT_NAME_MAX}	
9237	{_POSIX_TRACE_NAME_MAX}	
9238	{_POSIX_TRACE_SYS_MAX}	
9239	{_POSIX_TRACE_USER_EVENT_MAX}	
9240	{TRACE_EVENT_NAME_MAX}	
9241	{TRACE_NAME_MAX}	
9242	{TRACE_SYS_MAX}	
9243	{TRACE_USER_EVENT_MAX}	
9244	The new limits {_XOPEN_NAME_MAX} and {_XOPEN_PATH_MAX} are added as minimum	
9245	values for {PATH_MAX} and {NAME_MAX} limits on XSI-conformant systems.	
9246	The legacy symbols {PASS_MAX} and {TMP_MAX} are removed.	
9247	The values for the limits {CHAR_BIT}, {CHAR_MAX}, {SCHAR_MAX}, and {UCHAR_MAX} are	
9248	now required to be 8, +127, 255, and -128, respectively.	

9249 **NAME**

9250 locale.h — category macros

9251 **SYNOPSIS**

9252 #include <locale.h>

9253 **DESCRIPTION**

9254 **CX** Some of the functionality described on this reference page extends the ISO C standard. Any
 9255 conflict between the requirements described here and the ISO C standard is unintentional. This
 9256 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

9257 The **<locale.h>** header shall provide a definition for structure **lconv**, which shall include at least
 9258 the following members. (See the definitions of *LC_MONETARY* in the Section 7.3.3 (on page
 9259 138), and Section 7.3.4 (on page 141).)

```

9260 char    *currency_symbol
9261 char    *decimal_point
9262 char    frac_digits
9263 char    *grouping
9264 char    *int_curr_symbol
9265 char    int_frac_digits
9266 char    int_n_cs_precedes
9267 char    int_n_sep_by_space
9268 char    int_n_sign_posn
9269 char    int_p_cs_precedes
9270 char    int_p_sep_by_space
9271 char    int_p_sign_posn
9272 char    *mon_decimal_point
9273 char    *mon_grouping
9274 char    *mon_thousands_sep
9275 char    *negative_sign
9276 char    n_cs_precedes
9277 char    n_sep_by_space
9278 char    n_sign_posn
9279 char    *positive_sign
9280 char    p_cs_precedes
9281 char    p_sep_by_space
9282 char    p_sign_posn
9283 char    *thousands_sep

```

9284 The **<locale.h>** header shall define **NULL** (as defined in **<stddef.h>**) and at least the following as
 9285 macros:

```

9286 LC_ALL
9287 LC_COLLATE
9288 LC_CTYPE
9289 CX LC_MESSAGES
9290 LC_MONETARY
9291 LC_NUMERIC
9292 LC_TIME

```

9293 which shall expand to distinct integer constant expressions, for use as the first argument to the
 9294 *setlocale()* function.

9295 Additional macro definitions, beginning with the characters *LC_* and an uppercase letter, may
 9296 also be given here.

9297 The following shall be declared as functions and may also be defined as macros. Function |
9298 prototypes shall be provided. |

9299 struct lconv *localeconv (void); |
9300 char *setlocale(int, const char *); |

9301 **APPLICATION USAGE** |
9302 None. |

9303 **RATIONALE** |
9304 None. |

9305 **FUTURE DIRECTIONS** |
9306 None. |

9307 **SEE ALSO** |
9308 The System Interfaces volume of IEEE Std 1003.1-200x, *localeconv()*, *setlocale()*, Chapter 8 (on |
9309 page 157) |

9310 **CHANGE HISTORY** |
9311 First released in Issue 3. |
9312 Entry included for alignment with the ISO C standard. |

9313 **Issue 6** |
9314 The **lconv** structure is expanded with new members (**int_n_cs_precedes**, **int_n_sep_by_space**, |
9315 **int_n_sign_posn**, **int_p_cs_precedes**, **int_p_sep_by_space**, and **int_p_sign_posn**) for alignment |
9316 with the ISO/IEC 9899:1999 standard. |
9317 Extensions beyond the ISO C standard are now marked. |

9318 **NAME**9319 `math.h` — mathematical declarations9320 **SYNOPSIS**9321 `#include <math.h>`9322 **DESCRIPTION**

9323 `CX` Some of the functionality described on this reference page extends the ISO C standard.
 9324 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 9325 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these
 9326 symbols in this header.

9327 The **<math.h>** header shall include definitions for at least the following types: |9328 **float_t** A real-floating type at least as wide as **float**. |9329 **double_t** A real-floating type at least as wide as **double**, and at least as wide as **float_t**. |

9330 If `FLT_EVAL_METHOD` equals 0, **float_t** and **double_t** shall be **float** and **double**, respectively; if
 9331 `FLT_EVAL_METHOD` equals 1, they shall both be **double**; if `FLT_EVAL_METHOD` equals 2,
 9332 they shall both be **long double**; for other values of `FLT_EVAL_METHOD`, they are otherwise
 9333 implementation-defined.

9334 The **<math.h>** header shall define the following macros, where **real-floating** indicates that the
 9335 argument shall be an expression of real-floating type:

```
9336 int fpclassify(real-floating x);
9337 int isfinite(real-floating x);
9338 int isinf(real-floating x);
9339 int isnan(real-floating x);
9340 int isnormal(real-floating x);
9341 int signbit(real-floating x);
9342 int isgreater(real-floating x, real-floating y);
9343 int isgreaterequal(real-floating x, real-floating y);
9344 int isless(real-floating x, real-floating y);
9345 int islessequal(real-floating x, real-floating y);
9346 int islessgreater(real-floating x, real-floating y);
9347 int isunordered(real-floating x, real-floating y);
```

9348 The **<math.h>** header shall provide for the following constants. The values are of type **double**
 9349 and are accurate within the precision of the **double** type.

9350	<code>XSI</code>	<code>M_E</code>	Value of e
9351		<code>M_LOG2E</code>	Value of $\log_2 e$
9352		<code>M_LOG10E</code>	Value of $\log_{10} e$
9353		<code>M_LN2</code>	Value of $\log_e 2$
9354		<code>M_LN10</code>	Value of $\log_e 10$
9355		<code>M_PI</code>	Value of π
9356		<code>M_PI_2</code>	Value of $\pi/2$
9357		<code>M_PI_4</code>	Value of $\pi/4$
9358		<code>M_1_PI</code>	Value of $1/\pi$
9359		<code>M_2_PI</code>	Value of $2/\pi$

9360	<code>M_2_SQRTPI</code>	Value of $2\sqrt{\pi}$	
9361	<code>M_SQRT2</code>	Value of $\sqrt{2}$	
9362	<code>M_SQRT1_2</code>	Value of $1\sqrt{2}$	
9363	The header shall define the following symbolic constants:		
9364	<code>MAXFLOAT</code>	Value of maximum non-infinite single-precision floating-point number.	
9365	<code>HUGE_VAL</code>	A positive double expression, not necessarily representable as a float . Used as an error value returned by the mathematics library. <code>HUGE_VAL</code> evaluates to +infinity on systems supporting IEEE Std 754-1985.	
9366			
9367			
9368	<code>HUGE_VALF</code>	A positive float constant expression. Used as an error value returned by the mathematics library. <code>HUGE_VALF</code> evaluates to +infinity on systems supporting IEEE Std 754-1985.	
9369			
9370			
9371	<code>HUGE_VALL</code>	A positive long double constant expression. Used as an error value returned by the mathematics library. <code>HUGE_VALL</code> evaluates to +infinity on systems supporting IEEE Std 754-1985.	
9372			
9373			
9374	<code>INFINITY</code>	A constant expression of type float representing positive or unsigned infinity, if available; else a positive constant of type float that overflows at translation time.	
9375			
9376			
9377	<code>NAN</code>	A constant expression of type float representing a quiet NaN. This symbolic constant is only defined if the implementation supports quiet NaNs for the float type.	
9378			
9379			
9380	The following macros shall be defined for number classification. They represent the mutually-exclusive kinds of floating-point values. They expand to integer constant expressions with distinct values. Additional implementation-defined floating-point classifications, with macro definitions beginning with <code>FP_</code> and an uppercase letter, may also be specified by the implementation.		
9381			
9382			
9383			
9384			
9385	<code>FP_INFINITE</code>		
9386	<code>FP_NAN</code>		
9387	<code>FP_NORMAL</code>		
9388	<code>FP_SUBNORMAL</code>		
9389	<code>FP_ZERO</code>		
9390	The following optional macros indicate whether the <code>fma()</code> family of functions are fast compared with direct code:		
9391			
9392	<code>FP_FAST_FMA</code>		
9393	<code>FP_FAST_FMAF</code>		
9394	<code>FP_FAST_FMAL</code>		
9395	The <code>FP_FAST_FMA</code> macro shall be defined to indicate that the <code>fma()</code> function generally executes about as fast as, or faster than, a multiply and an add of double operands. The other macros have the equivalent meaning for the float and long double versions.		
9396			
9397			
9398	The following macros shall expand to integer constant expressions whose values are returned by <code>ilogb(x)</code> if <code>x</code> is zero or NaN, respectively. The value of <code>FP_ILOGB0</code> shall be either <code>{INT_MIN}</code> or <code>-{INT_MAX}</code> . The value of <code>FP_ILOGBNAN</code> shall be either <code>{INT_MAX}</code> or <code>{INT_MIN}</code> .		
9399			
9400			
9401	<code>FP_ILOGB0</code>		
9402	<code>FP_ILOGBNAN</code>		

9403 The following macros shall expand to the integer constants 1 and 2, respectively;

```
9404 MATH_ERRNO |
9405 MATH_ERREXCEPT |
```

9406 The following macro shall expand to an expression that has type **int** and the value
9407 MATH_ERRNO, MATH_ERREXCEPT, or the bitwise-inclusive OR of both:

```
9408 math_errhandling |
```

9409 The value of `math_errhandling` is constant for the duration of the program. It is unspecified |
9410 whether `math_errhandling` is a macro or an identifier with external linkage. If a macro definition |
9411 is suppressed or a program defines an identifier with the name `math_errhandling`, the behavior |
9412 is undefined. If the expression `(math_errhandling & MATH_ERREXCEPT)` can be non-zero, the |
9413 implementation shall define the macros `FE_DIVBYZERO`, `FE_INVALID`, and `FE_OVERFLOW` in |
9414 **<fenv.h>**.

9415 The following shall be declared as functions and may also be defined as macros. Function |
9416 prototypes shall be provided. |

```
9417 double      acos(double);
9418 float       acosf(float);
9419 double      acosh(double);
9420 float       acoshf(float);
9421 long double acoshl(long double);
9422 long double acosl(long double);
9423 double      asin(double);
9424 float       asinf(float);
9425 double      asinh(double);
9426 float       asinhf(float);
9427 long double asinhl(long double);
9428 long double asinl(long double);
9429 double      atan(double);
9430 double      atan2(double, double);
9431 float       atan2f(float, float);
9432 long double atan2l(long double, long double);
9433 float       atanf(float);
9434 double      atanh(double);
9435 float       atanhf(float);
9436 long double atanh1(long double);
9437 long double atanl(long double);
9438 double      cbrt(double);
9439 float       cbrtf(float);
9440 long double cbrtl(long double);
9441 double      ceil(double);
9442 float       ceilf(float);
9443 long double ceill(long double);
9444 double      copysign(double, double);
9445 float       copysignf(float, float);
9446 long double copysignl(long double, long double);
9447 double      cos(double);
9448 float       cosf(float);
9449 double      cosh(double);
9450 float       coshf(float);
9451 long double coshl(long double);
```

```

9452     long double cosl(long double);
9453     double      erf(double);
9454     double      erfc(double);
9455     float       erfcf(float);
9456     long double erfcl(long double);
9457     float       erff(float);
9458     long double erfl(long double);
9459     double      exp(double);
9460     double      exp2(double);
9461     float       exp2f(float);
9462     long double exp2l(long double);
9463     float       expf(float);
9464     long double expl(long double);
9465     double      expm1(double);
9466     float       expm1f(float);
9467     long double expm1l(long double);
9468     double      fabs(double);
9469     float       fabsf(float);
9470     long double fabsl(long double);
9471     double      fdim(double, double);
9472     float       fdimf(float, float);
9473     long double fdiml(long double, long double);
9474     double      floor(double);
9475     float       floorf(float);
9476     long double floorl(long double);
9477     double      fma(double, double, double);
9478     float       fmaf(float, float, float);
9479     long double fmal(long double, long double, long double);
9480     double      fmax(double, double);
9481     float       fmaxf(float, float);
9482     long double fmaxl(long double, long double);
9483     double      fmin(double, double);
9484     float       fminf(float, float);
9485     long double fminl(long double, long double);
9486     double      fmod(double, double);
9487     float       fmodf(float, float);
9488     long double fmodl(long double, long double);
9489     double      frexp(double, int *);
9490     float       frexpf(float value, int *);
9491     long double frexpl(long double value, int *);
9492     double      hypot(double, double);
9493     float       hypotf(float, float);
9494     long double hypotl(long double, long double);
9495     int         ilogb(double);
9496     int         ilogbf(float);
9497     int         ilogbl(long double);
9498 XSI    double   j0(double);
9499     double      j1(double);
9500     double      jn(int, double);
9501     double      ldexp(double, int);
9502     float       ldexpf(float, int);
9503     long double ldexpl(long double, int);

```

```
9504     double      lgamma(double);
9505     float        lgammaf(float);
9506     long double   lgammal(long double);
9507     long long     llrint(double);
9508     long long     llrintf(float);
9509     long long     llrintl(long double);
9510     long long     llround(double);
9511     long long     llroundf(float);
9512     long long     llroundl(long double);
9513     double        log(double);
9514     double        log10(double);
9515     float         log10f(float);
9516     long double   log10l(long double);
9517     double        log1p(double);
9518     float         log1pf(float);
9519     long double   log1pl(long double);
9520     double        log2(double);
9521     float         log2f(float);
9522     long double   log2l(long double);
9523     double        logb(double);
9524     float         logbf(float);
9525     long double   logbl(long double);
9526     float         logf(float);
9527     long double   logl(long double);
9528     long          lrint(double);
9529     long          lrintf(float);
9530     long          lrintl(long double);
9531     long          lround(double);
9532     long          lroundf(float);
9533     long          lroundl(long double);
9534     double        modf(double, double *);
9535     float         modff(float, float *);
9536     long double   modfl(long double, long double *);
9537     double        nan(const char *);
9538     float         nanf(const char *);
9539     long double   nanl(const char *);
9540     double        nearbyint(double);
9541     float         nearbyintf(float);
9542     long double   nearbyintl(long double);
9543     double        nextafter(double, double);
9544     float         nextafterf(float, float);
9545     long double   nextafterl(long double, long double);
9546     double        nexttoward(double, long double);
9547     float         nexttowardf(float, long double);
9548     long double   nexttowardl(long double, long double);
9549     double        pow(double, double);
9550     float         powf(float, float);
9551     long double   powl(long double, long double);
9552     double        remainder(double, double);
9553     float         remainderf(float, float);
9554     long double   remainderl(long double, long double);
9555     double        remquo(double, double, int *);
```

```

9556     float      remquof(float, float, int *);
9557     long double remquol(long double, long double, int *);
9558     double     rint(double);
9559     float      rintf(float);
9560     long double rintl(long double);
9561     double     round(double);
9562     float      roundf(float);
9563     long double roundl(long double);
9564 XSI    double     scalb(double, double);
9565     double     scalbln(double, long);
9566     float      scalblnf(float, long);
9567     long double scalblnl(long double, long);
9568     double     scalbn(double, int);
9569     float      scalbnf(float, int);
9570     long double scalbnl(long double, int);
9571     double     sin(double);
9572     float      sinf(float);
9573     double     sinh(double);
9574     float      sinhf(float);
9575     long double sinhl(long double);
9576     long double sinl(long double);
9577     double     sqrt(double);
9578     float      sqrtf(float);
9579     long double sqrtl(long double);
9580     double     tan(double);
9581     float      tanf(float);
9582     double     tanh(double);
9583     float      tanhf(float);
9584     long double tanhl(long double);
9585     long double tanl(long double);
9586     double     tgamma(double);
9587     float      tgammaf(float);
9588     long double tgammaL(long double);
9589     double     trunc(double);
9590     float      truncf(float);
9591     long double truncL(long double);
9592 XSI    double     y0(double);
9593     double     y1(double);
9594     double     yn(int, double);
9595

```

9596 The following external variable shall be defined:

```

9597 XSI    extern int signgam;
9598

```

9599 The behavior of each of the functions defined in <math.h> is specified in the System Interfaces
9600 volume of IEEE Std 1003.1-200x for all representable values of its input arguments, except where
9601 stated otherwise. Each function shall execute as if it were a single operation without generating
9602 any externally visible exceptional conditions.

9603 APPLICATION USAGE

9604 The FP_CONTRACT pragma can be used to allow (if the state is on) or disallow (if the state is
9605 off) the implementation to contract expressions. Each pragma can occur either outside external
9606 declarations or preceding all explicit declarations and statements inside a compound statement.
9607 When outside external declarations, the pragma takes effect from its occurrence until another
9608 FP_CONTRACT pragma is encountered, or until the end of the translation unit. When inside a
9609 compound statement, the pragma takes effect from its occurrence until another FP_CONTRACT
9610 pragma is encountered (including within a nested compound statement), or until the end of the
9611 compound statement; at the end of a compound statement the state for the pragma is restored to
9612 its condition just before the compound statement. If this pragma is used in any other context, the
9613 behavior is undefined. The default state (on or off) for the pragma is implementation-defined.

9614 RATIONALE

9615 Before the ISO/IEC 9899:1999 standard, the math library was defined only for the floating type
9616 **double**. All the names formed by appending 'f' or 'l' to a name in **<math.h>** were reserved
9617 to allow for the definition of **float** and **long double** libraries; and the ISO/IEC 9899:1999
9618 standard provides for all three versions of math functions.

9619 The functions *ecvt()*, *fcvt()*, and *gcvt()* have been dropped from the ISO C standard since their
9620 capability is available through *sprintf()*. These are provided on XSI-conformant systems
9621 supporting the Legacy Option Group.

9622 FUTURE DIRECTIONS

9623 None.

9624 SEE ALSO

9625 The System Interfaces volume of IEEE Std 1003.1-200x, *acos()*, *acosh()*, *asin()*, *atan()*, *atan2()*,
9626 *cbrt()*, *ceil()*, *cos()*, *cosh()*, *erf()*, *exp()*, *expm1()*, *fabs()*, *floor()*, *fmod()*, *frexp()*, *hypot()*, *ilogb()*,
9627 *isnan()*, *j0()*, *ldexp()*, *lgamma()*, *log()*, *log10()*, *log1p()*, *logb()*, *modf()*, *nextafter()*, *pow()*,
9628 *remainder()*, *rint()*, *scalb()*, *sin()*, *sinh()*, *sqrt()*, *tan()*, *tanh()*, *y0()*

9629 CHANGE HISTORY

9630 First released in Issue 1.

9631 Issue 6

9632 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

9633 **NAME**

9634 monetary.h — monetary types

9635 **SYNOPSIS**9636 XSI `#include <monetary.h>`

9637

9638 **DESCRIPTION**

9639 The <monetary.h> header shall define the following types:

9640 **size_t** As described in <stddef.h>.9641 **ssize_t** As described in <sys/types.h>.9642 The following shall be declared as a function and may also be defined as a macro. A function |
9643 prototype shall be provided. |9644 `ssize_t strfmon(char *restrict, size_t, const char *restrict, ...);`9645 **APPLICATION USAGE**

9646 None.

9647 **RATIONALE**

9648 None.

9649 **FUTURE DIRECTIONS**

9650 None.

9651 **SEE ALSO**9652 The System Interfaces volume of IEEE Std 1003.1-200x, *strfmon()*9653 **CHANGE HISTORY**

9654 First released in Issue 4.

9655 **Issue 6**9656 The **restrict** keyword is added to the prototype for *strfmon()*.

9657 **NAME**9658 mqueue.h — message queues (**REALTIME**)9659 **SYNOPSIS**

9660 MSG #include <mqueue.h>

9661

9662 **DESCRIPTION**9663 The <mqueue.h> header shall define the **mqd_t** type, which is used for message queue
9664 descriptors. This is not an array type.9665 The <mqueue.h> header shall define the **sigevent** structure (as described in <signal.h>) and the
9666 **mq_attr** structure, which is used in getting and setting the attributes of a message queue.
9667 Attributes are initially set when the message queue is created. An **mq_attr** structure shall have at
9668 least the following fields:

9669	long	mq_flags	Message queue flags.
9670	long	mq_maxmsg	Maximum number of messages.
9671	long	mq_msgsize	Maximum message size.
9672	long	mq_curmsgs	Number of messages currently queued.

9673 The following shall be declared as functions and may also be defined as macros. Function
9674 prototypes shall be provided.

```

9675 int      mq_close(mqd_t);
9676 int      mq_getattr(mqd_t, struct mq_attr *);
9677 int      mq_notify(mqd_t, const struct sigevent *);
9678 mqd_t    mq_open(const char *, int, ...);
9679 ssize_t  mq_receive(mqd_t, char *, size_t, unsigned *);
9680 int      mq_send(mqd_t, const char *, size_t, unsigned);
9681 int      mq_setattr(mqd_t, const struct mq_attr *restrict,
9682                  struct mq_attr *restrict);
9683 TMO     ssize_t  mq_timedreceive(mqd_t, char *restrict, size_t,
9684                                unsigned *restrict, const struct timespec *restrict);
9685 int      mq_timedsend(mqd_t, const char *, size_t, unsigned,
9686                      const struct timespec *);
9687 int      mq_unlink(const char *);

```

9688 Inclusion of the <mqueue.h> header may make visible symbols defined in the headers <fcntl.h>,
9689 <signal.h>, <sys/types.h>, and <time.h>.9690 **APPLICATION USAGE**

9691 None.

9692 **RATIONALE**

9693 None.

9694 **FUTURE DIRECTIONS**

9695 None.

9696 **SEE ALSO**9697 <fcntl.h>, <signal.h>, <sys/types.h>, <time.h>, the System Interfaces volume of
9698 IEEE Std 1003.1-200x, *mq_close()*, *mq_getattr()*, *mq_notify()*, *mq_open()*, *mq_receive()*, *mq_send()*,
9699 *mq_setattr()*, *mq_timedreceive()*, *mq_timedsend()*, *mq_unlink()*

9700 **CHANGE HISTORY**

9701 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

9702 **Issue 6**

9703 The <mqqueue.h> header is marked as part of the Message Passing option.

9704 The *mq_timedreceive()* and *mq_timedsend()* functions are added for alignment with
9705 IEEE Std 1003.1d-1999.

9706 The **restrict** keyword is added to the prototypes for *mq_setattr()* and *mq_timedreceive()*.

9707 **NAME**

9708 ndbm.h — definitions for ndbm database operations

9709 **SYNOPSIS**

9710 xSI #include <ndbm.h>

9711

9712 **DESCRIPTION**9713 The <ndbm.h> header shall define the **datum** type as a structure that includes at least the
9714 following members:

9715 void *dptr A pointer to the application's data.

9716 size_t dsize The size of the object pointed to by *dptr*.9717 The **size_t** type shall be defined as described in <stddef.h>.9718 The <ndbm.h> header shall define the **DBM** type.9719 The following constants shall be defined as possible values for the *store_mode* argument to
9720 *dbm_store()*:

9721 DBM_INSERT Insertion of new entries only.

9722 DBM_REPLACE Allow replacing existing entries.

9723 The following shall be declared as functions and may also be defined as macros. Function |
9724 prototypes shall be provided. |

9725 int dbm_clearerr(DBM *);

9726 void dbm_close(DBM *);

9727 int dbm_delete(DBM *, datum);

9728 int dbm_error(DBM *);

9729 datum dbm_fetch(DBM *, datum);

9730 datum dbm_firstkey(DBM *);

9731 datum dbm_nextkey(DBM *);

9732 DBM *dbm_open(const char *, int, mode_t);

9733 int dbm_store(DBM *, datum, datum, int);

9734 The **mode_t** type shall be defined through **typedef** as described in <sys/types.h>.9735 **APPLICATION USAGE**

9736 None.

9737 **RATIONALE**

9738 None.

9739 **FUTURE DIRECTIONS**

9740 None.

9741 **SEE ALSO**9742 The System Interfaces volume of IEEE Std 1003.1-200x, *dbm_clearerr()*9743 **CHANGE HISTORY**

9744 First released in Issue 4, Version 2.

9745 **Issue 5**9746 References to the definitions of **size_t** and **mode_t** are added to the DESCRIPTION.

9747 **NAME**

9748 net/if.h — sockets local interfaces

9749 **SYNOPSIS**

9750 #include <net/if.h>

9751 **DESCRIPTION**

9752 The <net/if.h> header shall define the **if_nameindex** structure that includes at least the
 9753 following members:

9754 unsigned if_index Numeric index of the interface.
 9755 char *if_name Null-terminated name of the interface.

9756 The <net/if.h> header shall define the following macro for the length of a buffer containing an
 9757 interface name (including the terminating NULL character):

9758 **IF_NAMESIZE** Interface name length.

9759 The following shall be declared as functions and may also be defined as macros. Function |
 9760 prototypes shall be provided. |

9761 unsigned if_nametoindex(const char *); |
 9762 char *if_indextoname(unsigned, char *); |
 9763 struct if_nameindex *if_nameindex(void); |
 9764 void if_freenameindex(struct if_nameindex *); |

9765 **APPLICATION USAGE** |

9766 None. |

9767 **RATIONALE**

9768 None.

9769 **FUTURE DIRECTIONS**

9770 None.

9771 **SEE ALSO**

9772 The System Interfaces volume of IEEE Std 1003.1-200x, *if_freenameindex()*, *if_indextoname()*,
 9773 *if_nameindex()*, *if_nametoindex()*

9774 **CHANGE HISTORY**

9775 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9776 **NAME**

9777 netdb.h — definitions for network database operations

9778 **SYNOPSIS**

9779 #include <netdb.h>

9780 **DESCRIPTION**9781 The <netdb.h> header may define the **in_port_t** type and the **in_addr_t** type as described in
9782 <netinet/in.h>.9783 The <netdb.h> header shall define the **hostent** structure that includes at least the following
9784 members:

9785	char	*h_name	Official name of the host.
9786	char	**h_aliases	A pointer to an array of pointers to 9787 alternative host names, terminated by a 9788 null pointer.
9789	int	h_addrtype	Address type.
9790	int	h_length	The length, in bytes, of the address.
9791	char	**h_addr_list	A pointer to an array of pointers to network 9792 addresses (in network byte order) for the host, 9793 terminated by a null pointer.

9794 The <netdb.h> header shall define the **netent** structure that includes at least the following
9795 members:

9796	char	*n_name	Official, fully-qualified (including the 9797 domain) name of the host.
9798	char	**n_aliases	A pointer to an array of pointers to 9799 alternative network names, terminated by a 9800 null pointer.
9801	int	n_addrtype	The address type of the network.
9802	uint32_t	n_net	The network number, in host byte order.

9803 The **uint32_t** type shall be defined as described in <inttypes.h>.9804 The <netdb.h> header shall define the **protoent** structure that includes at least the following
9805 members:

9806	char	*p_name	Official name of the protocol.
9807	char	**p_aliases	A pointer to an array of pointers to 9808 alternative protocol names, terminated by 9809 a null pointer.
9810	int	p_proto	The protocol number.

9811 The <netdb.h> header shall define the **servent** structure that includes at least the following
9812 members:

9813	char	*s_name	Official name of the service.
9814	char	**s_aliases	A pointer to an array of pointers to 9815 alternative service names, terminated by 9816 a null pointer.
9817	int	s_port	The port number at which the service 9818 resides, in network byte order.
9819	char	*s_proto	The name of the protocol to use when 9820 contacting the service.

9821 The <netdb.h> header shall define the IPPORT_RESERVED macro with the value of the highest
 9822 reserved Internet port number.

9823 OB When the <netdb.h> header is included, *h_errno* shall be available as a modifiable l-value of type
 9824 **int**. It is unspecified whether *h_errno* is a macro or an identifier declared with external linkage.

9825 The <netdb.h> header shall define the following macros for use as error values for
 9826 *gethostbyaddr()* and *gethostbyname()*:

9827 HOST_NOT_FOUND
 9828 NO_DATA
 9829 NO_RECOVERY
 9830 TRY_AGAIN

9831 **Address Information Structure**

9832 The <netdb.h> header shall define the **addrinfo** structure that includes at least the following
 9833 members:

9834	int	ai_flags	Input flags.
9835	int	ai_family	Address family of socket.
9836	int	ai_socktype	Socket type.
9837	int	ai_protocol	Protocol of socket.
9838	socklen_t	ai_addrlen	Length of socket address.
9839	struct sockaddr	*ai_addr	Socket address of socket.
9840	char	*ai_canonname	Canonical name of service location.
9841	struct addrinfo	*ai_next	Pointer to next in list.

9842 The <netdb.h> header shall define the following macros that evaluate to bitwise-distinct integer
 9843 constants for use in the *flags* field of the **addrinfo** structure:

9844	AI_PASSIVE	Socket address is intended for <i>bind()</i> .
9845	AI_CANONNAME	
9846		Request for canonical name.
9847	AI_NUMERICHOST	
9848		Return numeric host address as name.
9849	AI_NUMERICSERV	
9850		Inhibit service name resolution.
9851	AI_V4MAPPED	
9852		If no IPv6 addresses are found, query for IPv4 addresses and return them to
9853		the caller as IPv4-mapped IPv6 addresses.
9854	AI_ALL	Query for both IPv4 and IPv6 addresses.
9855	AI_ADDRCONFIG	
9856		Query for IPv4 addresses only when an IPv4 address is configured; query for
9857		IPv6 addresses only when an IPv6 address is configured.

9858 The <netdb.h> header shall define the following macros that evaluate to bitwise-distinct integer
 9859 constants for use in the *flags* argument to *getnameinfo()*:

9860	NI_NOFQDN	Only the nodename portion of the FQDN is returned for local hosts.
9861	NI_NUMERICHOST	
9862		The numeric form of the node's address is returned instead of its name.

9863 NI_NAMEREQD Return an error if the node's name cannot be located in the database.
 9864 NI_NUMERICSERV
 9865 The numeric form of the service address is returned instead of its name.
 9866 NI_DGRAM Indicates that the service is a datagram service (SOCK_DGRAM).

9867 **Address Information Errors**

9868 The <netdb.h> header shall define the following macros for use as error values for *getaddrinfo()*
 9869 and *getnameinfo()*:

9870 EAI_AGAIN The name could not be resolved at this time. Future attempts may succeed.
 9871 EAI_BADFLAGS The flags had an invalid value.
 9872 EAI_FAIL A non-recoverable error occurred.
 9873 EAI_FAMILY The address family was not recognized or the address length was invalid for
 9874 the specified family.
 9875 EAI_MEMORY There was a memory allocation failure.
 9876 EAI_NONAME The name does not resolve for the supplied parameters.
 9877 NI_NAMEREQD is set and the host's name cannot be located, or both
 9878 *nodename* and *servname* were null.

9879 EAI_SERVICE The service passed was not recognized for the specified socket type.

9880 EAI_SOCKTYPE The intended socket type was not recognized.

9881 EAI_SYSTEM A system error occurred. The error code can be found in *errno*. |

9882 EAI_OVERFLOW An argument buffer overflowed. |

9883 The following shall be declared as functions and may also be defined as macros. Function |
 9884 prototypes shall be provided. |

```

9885 void          endhostent(void);
9886 void          endnetent(void);
9887 void          endprotoent(void);
9888 void          endservent(void);
9889 void          freeaddrinfo(struct addrinfo *);
9890 const char    *gai_strerror(int);
9891 int           getaddrinfo(const char *restrict, const char *restrict, |
9892               const struct addrinfo *restrict, |
9893               struct addrinfo **restrict);
9894 struct hostent *gethostbyaddr(const void *, socklen_t, int);
9895 struct hostent *gethostbyname(const char *);
9896 struct hostent *gethostent(void);
9897 int           getnameinfo(const struct sockaddr *restrict, socklen_t, |
9898               char *restrict, socklen_t, char *restrict, |
9899               socklen_t, unsigned);
9900 struct netent *getnetbyaddr(uint32_t, int);
9901 struct netent *getnetbyname(const char *);
9902 struct netent *getnetent(void);
9903 struct protoent *getprotobyname(const char *);
9904 struct protoent *getprotobynumber(int);
9905 struct protoent *getprotoent(void);

```

```

9906     struct servent    *getservbyname(const char *, const char *);
9907     struct servent    *getservbyport(int, const char *);
9908     struct servent    *getservent(void);
9909     void              sethostent(int);
9910     void              setnetent(int);
9911     void              setprotoent(int);
9912     void              setservent(int);

```

9913 The type **socklen_t** shall be defined through **typedef** as described in <sys/socket.h>.

9914 Inclusion of the <netdb.h> header may also make visible all symbols from <netinet/in.h>, |
9915 <sys/socket.h>, and <inttypes.h>. |

9916 APPLICATION USAGE

9917 None.

9918 RATIONALE

9919 None.

9920 FUTURE DIRECTIONS

9921 None.

9922 SEE ALSO

9923 <netinet/in.h>, <inttypes.h>, <sys/socket.h>, the System Interfaces volume of
9924 IEEE Std 1003.1-200x, *bind()*, *endhostent()*, *endnetent()*, *endprotoent()*, *endservent()*, *getaddrinfo()*,
9925 *getnameinfo()*

9926 CHANGE HISTORY

9927 First released in Issue 6. Derived from the XNS, Issue 5.2 specification. |

9928 The Open Group Base Resolution bwg2001-009 is applied, which changes the return type for |
9929 *gai_strerror()* from **char *** to **const char ***. This is for coordination with the IPnG Working Group. |

9930 NAME

9931 netinet/in.h — Internet address family

9932 SYNOPSIS

9933 #include <netinet/in.h>

9934 DESCRIPTION

9935 The <netinet/in.h> header shall define the following types:

9936 **in_port_t** Equivalent to the type **uint16_t** as defined in <inttypes.h>.

9937 **in_addr_t** Equivalent to the type **uint32_t** as defined in <inttypes.h>.

9938 The **sa_family_t** type shall be defined as described in <sys/socket.h>.

9939 The **uint8_t** and **uint32_t** type shall be defined as described in <inttypes.h>. Inclusion of the
9940 <netinet/in.h> header may also make visible all symbols from <inttypes.h> and <sys/socket.h>.

9941 The <netinet/in.h> header shall define the **in_addr** structure that includes at least the following
9942 member:

9943 in_addr_t s_addr

9944 The <netinet/in.h> header shall define the **sockaddr_in** structure that includes at least the
9945 following members (all in network byte order):

9946 sa_family_t sin_family AF_INET.
9947 in_port_t sin_port Port number.
9948 struct in_addr sin_addr IP address.

9949 The **sockaddr_in** structure is used to store addresses for the Internet address family. Values of
9950 this type shall be cast by applications to **struct sockaddr** for use with socket functions.

9951 IP6 The <netinet/in.h> header shall define the **in6_addr** structure that contains at least the following
9952 member:

9953 uint8_t s6_addr[16]

9954 This array is used to contain a 128-bit IPv6 address, stored in network byte order.

9955 The <netinet/in.h> header shall define the **sockaddr_in6** structure that includes at least the
9956 following members (all in network byte order):

9957 sa_family_t sin6_family AF_INET6.
9958 in_port_t sin6_port Port number.
9959 uint32_t sin6_flowinfo IPv6 traffic class and flow information.
9960 struct in6_addr sin6_addr IPv6 address.
9961 uint32_t sin6_scope_id Set of interfaces for a scope.

9962 The **sockaddr_in6** structure shall be set to zero by an application prior to using it, since
9963 implementations are free to have additional, implementation-defined fields in **sockaddr_in6**.

9964 The *sin6_scope_id* field is a 32-bit integer that identifies a set of interfaces as appropriate for the
9965 scope of the address carried in the *sin6_addr* field. For a link scope *sin6_addr*, *sin6_scope_id* would
9966 be an interface index. For a site scope *sin6_addr*, *sin6_scope_id* would be a site identifier. The
9967 mapping of *sin6_scope_id* to an interface or set of interfaces is implementation-defined.

9968 The <netinet/in.h> header shall declare the following external variable:

9969 struct in6_addr in6addr_any

9970 This variable is initialized by the system to contain the wildcard IPv6 address. The
9971 <netinet/in.h> header also defines the IN6ADDR_ANY_INIT macro. This macro must be

9972 constant at compile time and can be used to initialize a variable of type **struct in6_addr** to the
 9973 IPv6 wildcard address.

9974 The <netinet/in.h> header shall declare the following external variable:

```
9975 struct in6_addr in6addr_loopback
```

9976 This variable is initialized by the system to contain the loopback IPv6 address. The
 9977 <netinet/in.h> header also defines the IN6ADDR_LOOPBACK_INIT macro. This macro must be
 9978 constant at compile time and can be used to initialize a variable of type **struct in6_addr** to the
 9979 IPv6 loopback address.

9980 The <netinet/in.h> header shall define the **ipv6_mreq** structure that includes at least the
 9981 following members:

```
9982 struct in6_addr  ipv6mr_multiaddr  IPv6 multicast address.  

9983 unsigned         ipv6mr_interface  Interface index.
```

9984

9985 The <netinet/in.h> header shall define the following macros for use as values of the *level*
 9986 argument of *getsockopt()* and *setsockopt()*:

9987	IPPROTO_IP	Internet protocol.
9988	IPPROTO_IPV6	Internet Protocol Version 6.
9989	IPPROTO_ICMP	Control message protocol.
9990	IPPROTO_RAW	Raw IP Packets Protocol.
9991	IPPROTO_TCP	Transmission control protocol.
9992	IPPROTO_UDP	User datagram protocol.

9993 The <netinet/in.h> header shall define the following macros for use as destination addresses for
 9994 *connect()*, *sendmsg()*, and *sendto()*:

9995	INADDR_ANY	IPv4 local host address.
9996	INADDR_BROADCAST	IPv4 broadcast address.

9997 The <netinet/in.h> header shall define the following macro to help applications declare buffers
 9998 of the proper size to store IPv4 addresses in string form:

9999	INET_ADDRSTRLEN	16. Length of the string form for IP.
------	-----------------	---------------------------------------

10000 The *htonl()*, *htons()*, *ntohl()*, and *ntohs()* functions shall be available as defined in <arpa/inet.h>.
 10001 Inclusion of the <netinet/in.h> header may also make visible all symbols from <arpa/inet.h>.

10002 IP6 The <netinet/in.h> header shall define the following macro to help applications declare buffers
 10003 of the proper size to store IPv6 addresses in string form:

10004	INET6_ADDRSTRLEN	46. Length of the string form for IPv6.
-------	------------------	---

10005 The <netinet/in.h> header shall define the following macros, with distinct integer values, for use
 10006 in the *option_name* argument in the *getsockopt()* or *setsockopt()* functions at protocol level
 10007 IPPROTO_IPV6:

10008	IPV6_JOIN_GROUP	Join a multicast group.
10009	IPV6_LEAVE_GROUP	Quit a multicast group.
10010	IPV6_MULTICAST_HOPS	
10011		Multicast hop limit.

10012	IPV6_MULTICAST_IF	Interface to use for outgoing multicast packets.
10013	IPV6_MULTICAST_LOOP	
10014		Multicast packets are delivered back to the local application.
10015	IPV6_UNICAST_HOPS	Unicast hop limit.
10016	IPV6_V6ONLY	Restrict AF_INET6 socket to IPv6 communications only.
10017	The <netinet/in.h> header shall define the following macros that test for special IPv6 addresses.	
10018	Each macro is of type int and takes a single argument of type const struct in6_addr* :	
10019	IN6_IS_ADDR_UNSPECIFIED	
10020		Unspecified address.
10021	IN6_IS_ADDR_LOOPBACK	
10022		Loopback address.
10023	IN6_IS_ADDR_MULTICAST	
10024		Multicast address.
10025	IN6_IS_ADDR_LINKLOCAL	
10026		Unicast link-local address.
10027	IN6_IS_ADDR_SITELOCAL	
10028		Unicast site-local address.
10029	IN6_IS_ADDR_V4MAPPED	
10030		IPv4 mapped address.
10031	IN6_IS_ADDR_V4COMPAT	
10032		IPv4-compatible address.
10033	IN6_IS_ADDR_MC_NODELOCAL	
10034		Multicast node-local address.
10035	IN6_IS_ADDR_MC_LINKLOCAL	
10036		Multicast link-local address.
10037	IN6_IS_ADDR_MC_SITELOCAL	
10038		Multicast site-local address.
10039	IN6_IS_ADDR_MC_ORGLOCAL	
10040		Multicast organization-local address.
10041	IN6_IS_ADDR_MC_GLOBAL	
10042		Multicast global address.
10043	IN6_IS_ADDR_LINKLOCAL and IN6_IS_ADDR_SITELOCAL return true only for the two	
10044	local-use IPv6 unicast addresses. They do not return true for multicast addresses of either link-	
10045	local or site-local scope.	

10046 **APPLICATION USAGE**

10047 None.

10048 **RATIONALE**

10049 None.

10050 **FUTURE DIRECTIONS**

10051 None.

10052 **SEE ALSO**

10053 Section 4.8 (on page 97), <arpa/inet.h>, <inttypes.h>, <sys/socket.h>, the System Interfaces |
10054 volume of IEEE Std 1003.1-200x, *connect()*, *getsockopt()*, *htonl()*, *htons()*, *ntohl()*, *ntohs()*, |
10055 *sendmsg()*, *sendto()*, *setsockopt()*

10056 **CHANGE HISTORY**

10057 First released in Issue 6. Derived from the XNS, Issue 5.2 specification. |

10058 The *sin_zero* member was removed from the **sockaddr_in** structure as per The Open Group Base |
10059 Resolution bwg2001-004. |

10060 **NAME**

10061 netinet/tcp.h — definitions for the Internet Transmission Control Protocol (TCP)

10062 **SYNOPSIS**

10063 #include <netinet/tcp.h>

10064 **DESCRIPTION**10065 The **<netinet/tcp.h>** header shall define the following macro for use as a socket option at the
10066 IPPROTO_TCP level:

10067 TCP_NODELAY Avoid coalescing of small segments.

10068 The macro shall be defined in the header. The implementation need not allow the value of the
10069 option to be set via *setsockopt()* or retrieved via *getsockopt()*.10070 **APPLICATION USAGE**

10071 None.

10072 **RATIONALE**

10073 None.

10074 **FUTURE DIRECTIONS**

10075 None.

10076 **SEE ALSO**10077 **<sys/socket.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *getsockopt()*, *setsockopt()*10078 **CHANGE HISTORY**

10079 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10080 **NAME**

10081 nl_types.h — data types

10082 **SYNOPSIS**

10083 XSI #include <nl_types.h>

10084

10085 **DESCRIPTION**

10086 The <nl_types.h> header shall contain definitions of at least the following types:

10087 **nl_catd** Used by the message catalog functions *catopen()*, *catgets()*, and *catclose()*
 10088 to identify a catalog descriptor.

10089 **nl_item** Used by *nl_langinfo()* to identify items of *langinfo* data. Values of objects
 10090 of type **nl_item** are defined in <langinfo.h>.

10091 The <nl_types.h> header shall contain definitions of at least the following constants:

10092 **NL_SETD** Used by *genocat* when no *\$set* directive is specified in a message text source
 10093 file; see the Internationalization Guide. This constant can be passed as the
 10094 value of *set_id* on subsequent calls to *catgets()* (that is, to retrieve
 10095 messages from the default message set). The value of **NL_SETD** is
 10096 implementation-defined.

10097 **NL_CAT_LOCALE** Value that must be passed as the *offlag* argument to *catopen()* to ensure
 10098 that message catalog selection depends on the *LC_MESSAGES* locale
 10099 category, rather than directly on the *LANG* environment variable.

10100 The following shall be declared as functions and may also be defined as macros. Function |
 10101 prototypes shall be provided. |

```
10102 int      catclose(nl_catd);
10103 char     *catgets(nl_catd, int, int, const char *);
10104 nl_catd  catopen(const char *, int);
```

10105 **APPLICATION USAGE**

10106 None.

10107 **RATIONALE**

10108 None.

10109 **FUTURE DIRECTIONS**

10110 None.

10111 **SEE ALSO**

10112 <langinfo.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *catclose()*, *catgets()*,
 10113 *catopen()*, *nl_langinfo()*, the Shell and Utilities volume of IEEE Std 1003.1-200x, *genocat*

10114 **CHANGE HISTORY**

10115 First released in Issue 2.

10116 NAME

10117 poll.h — definitions for the poll() function

10118 SYNOPSIS

10119 XSI #include <poll.h>

10120

10121 DESCRIPTION

10122 The <poll.h> header shall define the **pollfd** structure that includes at least the following
10123 members:

- 10124 int fd The following descriptor being polled.
- 10125 short events The input event flags (see below).
- 10126 short revents The output event flags (see below).

10127 The <poll.h> header shall define the following type through **typedef**:

10128 **nfds_t** An unsigned integer type used for the number of file descriptors.

10129 The implementation shall support one or more programming environments in which the width |
10130 of **nfds_t** is no greater than the width of type **long**. The names of these programming |
10131 environments can be obtained using the *confstr()* function or the *getconf* utility. |

10132 The following symbolic constants shall be defined, zero or more of which may be OR'ed together |
10133 to form the *events* or *revents* members in the **pollfd** structure:

- 10134 POLLIN Data other than high-priority data may be read without blocking.
- 10135 POLLRDNORM Normal data may be read without blocking.
- 10136 POLLRDBAND Priority data may be read without blocking.
- 10137 POLLPRI High priority data may be read without blocking.
- 10138 POLLOUT Normal data may be written without blocking.
- 10139 POLLWRNORM Equivalent to POLLOUT. |
- 10140 POLLWRBAND Priority data may be written.
- 10141 POLLERR An error has occurred (*revents* only).
- 10142 POLLHUP Device has been disconnected (*revents* only).
- 10143 POLLNVAL Invalid *fd* member (*revents* only).

10144 The significance and semantics of normal, priority, and high-priority data are file and device-
10145 specific.

10146 The following shall be declared as a function and may also be defined as a macro. A function |
10147 prototype shall be provided. |

10148 int poll(struct pollfd[], nfds_t, int);

10149 **APPLICATION USAGE**

10150 None.

10151 **RATIONALE**

10152 None.

10153 **FUTURE DIRECTIONS**

10154 None.

10155 **SEE ALSO**

10156 The System Interfaces volume of IEEE Std 1003.1-200x, *confstr()*, *poll()*, the Shell and Utilities |
10157 volume of IEEE Std 1003.1-200x, *getconf* |

10158 **CHANGE HISTORY**

10159 First released in Issue 4, Version 2.

10160 **Issue 6**10161 The description of the symbolic constants is updated to match the *poll()* function.10162 Text related to STREAMS has been moved to the *poll()* reference page.

10163 A note is added to the DESCRIPTION regarding the significance and semantics of normal,
10164 priority, and high-priority data.

10165 **NAME**

10166 pthread.h — threads

10167 **SYNOPSIS**

10168 THR #include <pthread.h>

10169

10170 **DESCRIPTION**

10171 The <pthread.h> header shall define the following symbols:

10172 BAR PTHREAD_BARRIER_SERIAL_THREAD

10173 PTHREAD_CANCEL_ASYNCHRONOUS

10174 PTHREAD_CANCEL_ENABLE

10175 PTHREAD_CANCEL_DEFERRED

10176 PTHREAD_CANCEL_DISABLE

10177 PTHREAD_CANCELED

10178 PTHREAD_COND_INITIALIZER

10179 PTHREAD_CREATE_DETACHED

10180 PTHREAD_CREATE_JOINABLE

10181 PTHREAD_EXPLICIT_SCHED

10182 PTHREAD_INHERIT_SCHED

10183 XSI PTHREAD_MUTEX_DEFAULT

10184 PTHREAD_MUTEX_ERRORCHECK

10185 PTHREAD_MUTEX_INITIALIZER

10186 XSI PTHREAD_MUTEX_NORMAL

10187 PTHREAD_MUTEX_RECURSIVE

10188 PTHREAD_ONCE_INIT

10189 TPP|TPI PTHREAD_PRIO_INHERIT

10190 PTHREAD_PRIO_NONE

10191 PTHREAD_PRIO_PROTECT

10192 PTHREAD_PROCESS_SHARED

10193 PTHREAD_PROCESS_PRIVATE

10194 TPS PTHREAD_SCOPE_PROCESS

10195 PTHREAD_SCOPE_SYSTEM

10196

10197 The following types shall be defined as described in <sys/types.h>:

10198 pthread_attr_t

10199 BAR pthread_barrier_t

10200 pthread_barrierattr_t

10201 pthread_cond_t

10202 pthread_condattr_t

10203 pthread_key_t

10204 pthread_mutex_t

10205 pthread_mutexattr_t

10206 pthread_once_t

10207 pthread_rwlock_t

10208 pthread_rwlockattr_t

10209 SPI pthread_spinlock_t

10210 pthread_t

10211 The following shall be declared as functions and may also be defined as macros. Function |
10212 prototypes shall be provided. |

```

10213     int    pthread_atfork(void (*)(void), void (*)(void),
10214                          void (*)(void));
10215     int    pthread_attr_destroy(pthread_attr_t *);
10216     int    pthread_attr_getdetachstate(const pthread_attr_t *, int *);
10217 XSI    int    pthread_attr_getguardsize(const pthread_attr_t *restrict,
10218                          size_t *restrict);
10219 TPS    int    pthread_attr_getinheritsched(const pthread_attr_t *restrict,
10220                          int *restrict);
10221     int    pthread_attr_getschedparam(const pthread_attr_t *restrict,
10222                          struct sched_param *restrict);
10223 TPS    int    pthread_attr_getschedpolicy(const pthread_attr_t *restrict,
10224                          int *restrict);
10225 TPS    int    pthread_attr_getscope(const pthread_attr_t *restrict,
10226                          int *restrict);
10227 XSI    int    pthread_attr_getstack(const pthread_attr_t *restrict,
10228                          void **restrict, size_t *restrict);
10229 TSA    int    pthread_attr_getstackaddr(const pthread_attr_t *restrict,
10230                          void **restrict);
10231     int    pthread_attr_getstacksize(const pthread_attr_t *restrict,
10232                          size_t *restrict);
10233     int    pthread_attr_init(pthread_attr_t *);
10234     int    pthread_attr_setdetachstate(pthread_attr_t *, int);
10235 XSI    int    pthread_attr_setguardsize(pthread_attr_t *, size_t);
10236 TPS    int    pthread_attr_setinheritsched(pthread_attr_t *, int);
10237     int    pthread_attr_setschedparam(pthread_attr_t *restrict,
10238                          const struct sched_param *restrict);
10239 TPS    int    pthread_attr_setschedpolicy(pthread_attr_t *, int);
10240     int    pthread_attr_setscope(pthread_attr_t *, int);
10241 XSI    int    pthread_attr_setstack(pthread_attr_t *, void *, size_t);
10242 TSA    int    pthread_attr_setstackaddr(pthread_attr_t *, void *);
10243     int    pthread_attr_setstacksize(pthread_attr_t *, size_t);
10244 BAR    int    pthread_barrier_destroy(pthread_barrier_t *);
10245     int    pthread_barrier_init(pthread_barrier_t *restrict,
10246                          const pthread_barrierattr_t *restrict, unsigned);
10247     int    pthread_barrier_wait(pthread_barrier_t *);
10248     int    pthread_barrierattr_destroy(pthread_barrierattr_t *);
10249     int    pthread_barrierattr_getpshared( \
10250                          const pthread_barrierattr_t *restrict, int *restrict);
10251     int    pthread_barrierattr_init(pthread_barrierattr_t *);
10252     int    pthread_barrierattr_setpshared(pthread_barrierattr_t *, int);
10253     int    pthread_cancel(pthread_t);
10254     void   pthread_cleanup_push(void (*)(void *), void *);
10255     void   pthread_cleanup_pop(int);
10256     int    pthread_cond_broadcast(pthread_cond_t *);
10257     int    pthread_cond_destroy(pthread_cond_t *);
10258     int    pthread_cond_init(pthread_cond_t *restrict,
10259                          const pthread_condattr_t *restrict);
10260     int    pthread_cond_signal(pthread_cond_t *);
10261     int    pthread_cond_timedwait(pthread_cond_t *restrict,
10262                          pthread_mutex_t *restrict, const struct timespec *restrict);
10263     int    pthread_cond_wait(pthread_cond_t *restrict,
10264                          pthread_mutex_t *restrict);

```

```

10265     int  pthread_condattr_destroy(pthread_condattr_t *);
10266 CS   int  pthread_condattr_getclock(const pthread_condattr_t *restrict,
10267     clockid_t *restrict);
10268     int  pthread_condattr_getpshared(const pthread_condattr_t *restrict,
10269     int *restrict);
10270     int  pthread_condattr_init(pthread_condattr_t *);
10271 CS   int  pthread_condattr_setclock(pthread_condattr_t *, clockid_t);
10272     int  pthread_condattr_setpshared(pthread_condattr_t *, int);
10273     int  pthread_create(pthread_t *restrict, const pthread_attr_t *restrict,
10274     void *(*)(void *), void *restrict);
10275     int  pthread_detach(pthread_t);
10276     int  pthread_equal(pthread_t, pthread_t);
10277     void pthread_exit(void *);
10278 XSI   int  pthread_getconcurrency(void);
10279 TCT   int  pthread_getcpuclockid(pthread_t, clockid_t *);
10280 TPS   int  pthread_getschedparam(pthread_t, int *restrict,
10281     struct sched_param *restrict);
10282     void *pthread_getspecific(pthread_key_t);
10283     int  pthread_join(pthread_t, void **);
10284     int  pthread_key_create(pthread_key_t *, void (*)(void *));
10285     int  pthread_key_delete(pthread_key_t);
10286     int  pthread_mutex_destroy(pthread_mutex_t *);
10287 TPP   int  pthread_mutex_getprioceiling(const pthread_mutex_t *restrict,
10288     int *restrict);
10289     int  pthread_mutex_init(pthread_mutex_t *restrict,
10290     const pthread_mutexattr_t *restrict);
10291     int  pthread_mutex_lock(pthread_mutex_t *);
10292 TPP   int  pthread_mutex_setprioceiling(pthread_mutex_t *restrict, int,
10293     int *restrict);
10294 TMO   int  pthread_mutex_timedlock(pthread_mutex_t *,
10295     const struct timespec *);
10296     int  pthread_mutex_trylock(pthread_mutex_t *);
10297     int  pthread_mutex_unlock(pthread_mutex_t *);
10298     int  pthread_mutexattr_destroy(pthread_mutexattr_t *);
10299 TPP|TPI int  pthread_mutexattr_getprioceiling(\
10300     const pthread_mutexattr_t *restrict, int *restrict);
10301     int  pthread_mutexattr_getprotocol(const pthread_mutexattr_t *restrict,
10302     int *restrict);
10303     int  pthread_mutexattr_getpshared(const pthread_mutexattr_t *restrict,
10304     int *restrict);
10305 XSI   int  pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict,
10306     int *restrict);
10307     int  pthread_mutexattr_init(pthread_mutexattr_t *);
10308 TPP|TPI int  pthread_mutexattr_setprioceiling(pthread_mutexattr_t *, int);
10309     int  pthread_mutexattr_setprotocol(pthread_mutexattr_t *, int);
10310     int  pthread_mutexattr_setpshared(pthread_mutexattr_t *, int);
10311 XSI   int  pthread_mutexattr_settype(pthread_mutexattr_t *, int);
10312     int  pthread_once(pthread_once_t *, void (*)(void));
10313     int  pthread_rwlock_destroy(pthread_rwlock_t *);
10314     int  pthread_rwlock_init(pthread_rwlock_t *restrict,
10315     const pthread_rwlockattr_t *restrict);
10316     int  pthread_rwlock_rdlock(pthread_rwlock_t *);

```

```

10317     int    pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict,
10318         const struct timespec *restrict);
10319     int    pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict,
10320         const struct timespec *restrict);
10321     int    pthread_rwlock_tryrdlock(pthread_rwlock_t *);
10322     int    pthread_rwlock_trywrlock(pthread_rwlock_t *);
10323     int    pthread_rwlock_unlock(pthread_rwlock_t *);
10324     int    pthread_rwlock_wrlock(pthread_rwlock_t *);
10325     int    pthread_rwlockattr_destroy(pthread_rwlockattr_t *);
10326     int    pthread_rwlockattr_getpshared(const pthread_rwlockattr_t *restrict,
10327         int *restrict);
10328     int    pthread_rwlockattr_init(pthread_rwlockattr_t *);
10329     int    pthread_rwlockattr_setpshared(pthread_rwlockattr_t *, int);
10330     pthread_t
10331         pthread_self(void);
10332     int    pthread_setcancelstate(int, int *);
10333     int    pthread_setcanceltype(int, int *);
10334 XSI     int    pthread_setconcurrency(int);
10335 TPS     int    pthread_setschedparam(pthread_t, int,
10336         const struct sched_param *);
10337 THR TPS int    pthread_setschedprio(pthread_t, int);
10338     int    pthread_setspecific(pthread_key_t, const void *);
10339 SPI     int    pthread_spin_destroy(pthread_spinlock_t *);
10340     int    pthread_spin_init(pthread_spinlock_t *, int);
10341     int    pthread_spin_lock(pthread_spinlock_t *);
10342     int    pthread_spin_trylock(pthread_spinlock_t *);
10343     int    pthread_spin_unlock(pthread_spinlock_t *);
10344     void   pthread_testcancel(void);

```

10345 Inclusion of the <pthread.h> header shall make symbols defined in the headers <sched.h> and
10346 <time.h> visible.

10347 **APPLICATION USAGE**

10348 None.

10349 **RATIONALE**

10350 None.

10351 **FUTURE DIRECTIONS**

10352 None.

10353 **SEE ALSO**

10354 <sched.h>, <time.h>, the System Interfaces volume of IEEE Std 1003.1-200x,
10355 *pthread_attr_getguardsize()*, *pthread_attr_init()*, *pthread_attr_setscope()*, *pthread_barrier_destroy()*,
10356 *pthread_barrier_init()*, *pthread_barrier_wait()*, *pthread_barrierattr_destroy()*,
10357 *pthread_barrierattr_getpshared()*, *pthread_barrierattr_init()*, *pthread_barrierattr_setpshared()*,
10358 *pthread_cancel()*, *pthread_cleanup_pop()*, *pthread_cond_init()*, *pthread_cond_signal()*,
10359 *pthread_cond_wait()*, *pthread_condattr_getclock()*, *pthread_condattr_init()*,
10360 *pthread_condattr_setclock()*, *pthread_create()*, *pthread_detach()*, *pthread_equal()*, *pthread_exit()*,
10361 *pthread_getconcurrency()*, *pthread_getcpuclockid()*, *pthread_getschedparam()*, *pthread_join()*,
10362 *pthread_key_create()*, *pthread_key_delete()*, *pthread_mutex_init()*, *pthread_mutex_lock()*,
10363 *pthread_mutex_setprioceiling()*, *pthread_mutex_timedlock()*, *pthread_mutexattr_init()*,
10364 *pthread_mutexattr_gettype()*, *pthread_mutexattr_setprotocol()*, *pthread_once()*,
10365 *pthread_rwlock_destroy()*, *pthread_rwlock_init()*, *pthread_rwlock_rdlock()*,
10366 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_rwlock_tryrdlock()*,

10367 *pthread_rwlock_trywrlock()*, *pthread_rwlock_unlock()*, *pthread_rwlock_wrlock()*,
 10368 *pthread_rwlockattr_destroy()*, *pthread_rwlockattr_getpshared()*, *pthread_rwlockattr_init()*,
 10369 *pthread_rwlockattr_setpshared()*, *pthread_self()*, *pthread_setcancelstate()*, *pthread_setspecific()*,
 10370 *pthread_spin_destroy()*, *pthread_spin_init()*, *pthread_spin_lock()*, *pthread_spin_trylock()*,
 10371 *pthread_spin_unlock()*

10372 **CHANGE HISTORY**

10373 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

10374 **Issue 6**

10375 The RTT margin markers are now broken out into their POSIX options.

10376 The Open Group Corrigendum U021/9 is applied, correcting the prototype for the
 10377 *pthread_cond_wait()* function.

10378 The Open Group Corrigendum U026/2 is applied correcting the prototype for the
 10379 *pthread_setschedparam()* function so that its second argument is of type **int**.

10380 The *pthread_getcpuclockid()* and *pthread_mutex_timedlock()* functions are added for alignment
 10381 with IEEE Std 1003.1d-1999.

10382 The following functions are added for alignment with IEEE Std 1003.1j-2000:

10383 *pthread_barrier_destroy()*, *pthread_barrier_init()*, *pthread_barrier_wait()*,
 10384 *pthread_barrierattr_destroy()*, *pthread_barrierattr_getpshared()*, *pthread_barrierattr_init()*,
 10385 *pthread_barrierattr_setpshared()*, *pthread_condattr_getclock()*, *pthread_condattr_setclock()*,
 10386 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_spin_destroy()*,
 10387 *pthread_spin_init()*, *pthread_spin_lock()*, *pthread_spin_trylock()*, and *pthread_spin_unlock()*.

10388 PTHREAD_RWLOCK_INITIALIZER is deleted for alignment with IEEE Std 1003.1j-2000.

10389 Functions previously marked as part of the Read-Write Locks option are now moved to the
 10390 Threads option.

10391 The **restrict** keyword is added to the prototypes for *pthread_attr_getguardsize()*,
 10392 *pthread_attr_getinheritsched()*, *pthread_attr_getschedparam()*, *pthread_attr_getschedpolicy()*,
 10393 *pthread_attr_getscope()*, *pthread_attr_getstackaddr()*, *pthread_attr_getstacksize()*,
 10394 *pthread_attr_setschedparam()*, *pthread_barrier_init()*, *pthread_barrierattr_getpshared()*,
 10395 *pthread_cond_init()*, *pthread_cond_signal()*, *pthread_cond_timedwait()*, *pthread_cond_wait()*,
 10396 *pthread_condattr_getclock()*, *pthread_condattr_getpshared()*, *pthread_create()*,
 10397 *pthread_getschedparam()*, *pthread_mutex_getprioceiling()*, *pthread_mutex_init()*,
 10398 *pthread_mutex_setprioceiling()*, *pthread_mutexattr_getprioceiling()*, *pthread_mutexattr_getprotocol()*,
 10399 *pthread_mutexattr_getpshared()*, *pthread_mutexattr_gettype()*, *pthread_rwlock_init()*,
 10400 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_rwlockattr_getpshared()*, and
 10401 *pthread_sigmask()*.

10402 IEEE PASC Interpretation 1003.1 #86 is applied, allowing the symbols from <sched.h> and
 10403 <time.h> to be made visible when <pthread.h> is included. Previously this was an XSI
 10404 extension.

10405 IEEE PASC Interpretation 1003.1c #42 is applied, removing the requirement for prototypes for
 10406 the *pthread_kill()* and *pthread_sigmask()* functions. These are required to be in the <signal.h>
 10407 header. They are allowed here through the name space rules. |

10408 IEEE PASC Interpretation 1003.1 #96 is applied, adding the *pthread_setschedprio()* function. |

10409 **NAME**

10410 pwd.h — password structure

10411 **SYNOPSIS**

10412 #include <pwd.h>

10413 **DESCRIPTION**

10414 The <pwd.h> header shall provide a definition for **struct passwd**, which shall include at least the
 10415 following members:

10416	char	*pw_name	User's login name.
10417	uid_t	pw_uid	Numerical user ID.
10418	gid_t	pw_gid	Numerical group ID.
10419	char	*pw_dir	Initial working directory.
10420	char	*pw_shell	Program to use as shell.

10421 The **gid_t** and **uid_t** types shall be defined as described in <sys/types.h>.

10422 The following shall be declared as functions and may also be defined as macros. Function
 10423 prototypes shall be provided.

```

10424 struct passwd *getpwnam(const char *);
10425 struct passwd *getpwuid(uid_t);
10426 TSF int getpwnam_r(const char *, struct passwd *, char *,
10427 size_t, struct passwd **);
10428 int getpwuid_r(uid_t, struct passwd *, char *,
10429 size_t, struct passwd **);
10430 XSI void endpwent(void);
10431 struct passwd *getpwent(void);
10432 void setpwent(void);
    
```

10434 **APPLICATION USAGE**

10435 None.

10436 **RATIONALE**

10437 None.

10438 **FUTURE DIRECTIONS**

10439 None.

10440 **SEE ALSO**

10441 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *endpwent()*, *getpwnam()*,
 10442 *getpwuid()*

10443 **CHANGE HISTORY**

10444 First released in Issue 1.

10445 **Issue 5**

10446 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

10447 **Issue 6**

10448 The following new requirements on POSIX implementations derive from alignment with the
 10449 Single UNIX Specification:

- 10450 • The **gid_t** and **uid_t** types are mandated.
- 10451 • The *getpwnam_r()* and *getpwuid_r()* functions are marked as part of the
 10452 `_POSIX_THREAD_SAFE_FUNCTIONS` option.

10453 **NAME**10454 `regex.h` — regular expression matching types10455 **SYNOPSIS**10456 `#include <regex.h>`10457 **DESCRIPTION**10458 The **<regex.h>** header shall define the structures and symbolic constants used by the *regcomp()*,
10459 *regexec()*, *regerror()*, and *regfree()* functions.10460 The structure type **regex_t** shall contain at least the following member:10461 `size_t re_nsub` Number of parenthesized subexpressions.10462 The type **size_t** shall be defined as described in **<sys/types.h>**.10463 The type **regoff_t** shall be defined as a signed integer type that can hold the largest value that
10464 can be stored in either a type **off_t** or type **ssize_t**. The structure type **regmatch_t** shall contain
10465 at least the following members:10466 `regoff_t rm_so` Byte offset from start of string
10467 to start of substring.
10468 `regoff_t rm_eo` Byte offset from start of string of the
10469 first character after the end of substring.10470 Values for the *cflags* parameter to the *regcomp()* function:10471 **REG_EXTENDED** Use Extended Regular Expressions.10472 **REG_ICASE** Ignore case in match.10473 **REG_NOSUB** Report only success or fail in *regexec()*.10474 **REG_NEWLINE** Change the handling of newline.10475 Values for the *eflags* parameter to the *regexec()* function:10476 **REG_NOTBOL** The circumflex character ('^'), when taken as a special character, does
10477 not match the beginning of *string*.10478 **REG_NOTEOL** The dollar sign ('\$'), when taken as a special character, does not match
10479 the end of *string*.

10480 The following constants shall be defined as error return values:

10481 **REG_NOMATCH** *regexec()* failed to match.10482 **REG_BADPAT** Invalid regular expression.10483 **REG_ECOLLATE** Invalid collating element referenced.10484 **REG_ECTYPE** Invalid character class type referenced.10485 **REG_EESCAPE** Trailing '\\' in pattern.10486 **REG_ESUBREG** Number in *\digit* invalid or in error.10487 **REG_EBRACK** "[]" imbalance.10488 **REG_EPAREN** "\(\)" or "()" imbalance.10489 **REG_EBRACE** "\{\}" imbalance.10490 **REG_BADBR** Content of "\{\}" invalid: not a number, number too large, more than
10491 two numbers, first larger than second.

10492	REG_ERANGE	Invalid endpoint in range expression.
10493	REG_ESPACE	Out of memory.
10494	REG_BADRPT	'?', '*', or '+' not preceded by valid regular expression.
10495	OB REG_ENOSYS	Reserved.
10496		The following shall be declared as functions and may also be defined as macros. Function
10497		prototypes shall be provided.
10498		<code>int regcomp(regex_t *restrict, const char *restrict, int);</code>
10499		<code>size_t regerror(int, const regex_t *restrict, char *restrict, size_t);</code>
10500		<code>int regexec(const regex_t *restrict, const char *restrict, size_t,</code>
10501		<code>regmatch_t[restrict], int);</code>
10502		<code>void regfree(regex_t *);</code>
10503		The implementation may define additional macros or constants using names beginning with
10504		REG_.
10505	APPLICATION USAGE	
10506		None.
10507	RATIONALE	
10508		None.
10509	FUTURE DIRECTIONS	
10510		None.
10511	SEE ALSO	
10512		The System Interfaces volume of IEEE Std 1003.1-200x, <i>regcomp()</i> , the Shell and Utilities volume
10513		of IEEE Std 1003.1-200x
10514	CHANGE HISTORY	
10515		First released in Issue 4.
10516		Originally derived from the ISO POSIX-2 standard.
10517	Issue 6	
10518		The REG_ENOSYS constant is marked obsolescent.
10519		The restrict keyword is added to the prototypes for <i>regcomp()</i> , <i>regerror()</i> , and <i>regexec()</i> .
10520		A statement is added that the size_t type is defined as described in <sys/types.h>.

10521 NAME

10522 sched.h — execution scheduling (**REALTIME**)

10523 SYNOPSIS

10524 PS #include <sched.h>

10525

10526 DESCRIPTION

10527 The <sched.h> header shall define the **sched_param** structure, which contains the scheduling
10528 parameters required for implementation of each supported scheduling policy. This structure
10529 shall contain at least the following member:

10530 int sched_priority Process execution scheduling priority.

10531 SS|TSP In addition, if **_POSIX_SPORADIC_SERVER** or **_POSIX_THREAD_SPORADIC_SERVER** is
10532 defined, the **sched_param** structure defined in <sched.h> shall contain the following members
10533 in addition to those specified above:

10534 int sched_ss_low_priority Low scheduling priority for
10535 sporadic server.
10536 struct timespec sched_ss_repl_period Replenishment period for
10537 sporadic server.
10538 struct timespec sched_ss_init_budget Initial budget for sporadic server.
10539 int sched_ss_max_repl Maximum pending replenishments for
10540 sporadic server.

10541

10542 Each process is controlled by an associated scheduling policy and priority. Associated with each
10543 policy is a priority range. Each policy definition specifies the minimum priority range for that
10544 policy. The priority ranges for each policy may overlap the priority ranges of other policies.

10545 Four scheduling policies are defined; others may be defined by the implementation. The four
10546 standard policies are indicated by the values of the following symbolic constants:

10547 **SCHED_FIFO** First in-first out (FIFO) scheduling policy.

10548 **SCHED_RR** Round robin scheduling policy.

10549 SS|TSP **SCHED_SPORADIC** Sporadic server scheduling policy.

10550 **SCHED_OTHER** Another scheduling policy.

10551 The values of these constants are distinct.

10552 The following shall be declared as functions and may also be defined as macros. Function |
10553 prototypes shall be provided. |

10554 int sched_get_priority_max(int);
10555 int sched_get_priority_min(int);
10556 int sched_getparam(pid_t, struct sched_param *);
10557 int sched_getscheduler(pid_t);
10558 int sched_rr_get_interval(pid_t, struct timespec *);
10559 int sched_setparam(pid_t, const struct sched_param *);
10560 int sched_setscheduler(pid_t, int, const struct sched_param *);
10561 int sched_yield(void);

10562 Inclusion of the <sched.h> header makes symbols defined in the header <time.h> visible.

10563 **APPLICATION USAGE**

10564 None.

10565 **RATIONALE**

10566 None.

10567 **FUTURE DIRECTIONS**

10568 None.

10569 **SEE ALSO**

10570 <time.h>

10571 **CHANGE HISTORY**

10572 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

10573 **Issue 6**

10574 The <sched.h> header is marked as part of the Process Scheduling option.

10575 Sporadic server members are added to the **sched_param** structure, and the SCHED_SPORADIC
10576 scheduling policy is added for alignment with IEEE Std 1003.1d-1999.10577 IEEE PASC Interpretation 1003.1 #108 is applied, correcting the **sched_param** structure whose
10578 members *sched_ss_repl_period* and *sched_ss_init_budget* members should be type **struct timespec**
10579 and not **timespec**.

10580 **NAME**

10581 search.h — search tables

10582 **SYNOPSIS**

10583 XSI #include <search.h>

10584

10585 **DESCRIPTION**

10586 The **<search.h>** header shall define the **ENTRY** type for structure **entry** which shall include the
 10587 following members:

```
10588 char    *key
10589 void    *data
```

10590 and shall define **ACTION** and **VISIT** as enumeration data types through type definitions as
 10591 follows:

```
10592 enum { FIND, ENTER } ACTION;
10593 enum { preorder, postorder, endorder, leaf } VISIT;
```

10594 The **size_t** type shall be defined as described in **<sys/types.h>**.

10595 The following shall be declared as functions and may also be defined as macros. Function |
 10596 prototypes shall be provided. |

```
10597 int    hcreate(size_t);
10598 void   hdestroy(void);
10599 ENTRY *hsearch(ENTRY, ACTION);
10600 void   insque(void *, void *);
10601 void   *lfind(const void *, const void *, size_t *,
10602             size_t, int (*)(const void *, const void *));
10603 void   *lsearch(const void *, void *, size_t *,
10604               size_t, int (*)(const void *, const void *));
10605 void   remque(void *);
10606 void   *tdelete(const void *restrict, void **restrict,
10607                int (*)(const void *, const void *));
10608 void   *tfind(const void *, void *const *,
10609              int (*)(const void *, const void *));
10610 void   *tsearch(const void *, void **,
10611                int (*)(const void *, const void *));
10612 void   twalk(const void *,
10613             void (*)(const void *, VISIT, int ));
```

10614 **APPLICATION USAGE**

10615 None.

10616 **RATIONALE**

10617 None.

10618 **FUTURE DIRECTIONS**

10619 None.

10620 **SEE ALSO**

10621 **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *hcreate()*, *insque()*,
 10622 *lsearch()*, *remque()*, *tsearch()*

10623 **CHANGE HISTORY**

10624 First released in Issue 1. Derived from Issue 1 of the SVID.

10625 **Issue 6**

10626 The Open Group Corrigendum U021/6 is applied updating the prototypes for *tdelete()* and
10627 *tsearch()*.

10628 The **restrict** keyword is added to the prototype for *tdelete()*.

10629 **NAME**10630 semaphore.h — semaphores (**REALTIME**)10631 **SYNOPSIS**10632 SEM `#include <semaphore.h>`

10633

10634 **DESCRIPTION**

10635 The **<semaphore.h>** header shall define the **sem_t** type, used in performing semaphore
10636 operations. The semaphore may be implemented using a file descriptor, in which case
10637 applications are able to open up at least a total of {OPEN_MAX} files and semaphores. The
10638 symbol SEM_FAILED shall be defined (see *sem_open()*).

10639 The following shall be declared as functions and may also be defined as macros. Function
10640 prototypes shall be provided.

10641 `int sem_close(sem_t *);`10642 `int sem_destroy(sem_t *);`10643 `int sem_getvalue(sem_t *restrict, int *restrict);`10644 `int sem_init(sem_t *, int, unsigned);`10645 `sem_t *sem_open(const char *, int, ...);`10646 `int sem_post(sem_t *);`10647 TMO `int sem_timedwait(sem_t *restrict, const struct timespec *restrict);`10648 `int sem_trywait(sem_t *);`10649 `int sem_unlink(const char *);`10650 `int sem_wait(sem_t *);`

10651 Inclusion of the **<semaphore.h>** header may make visible symbols defined in the headers
10652 **<fcntl.h>** and **<sys/types.h>**.

10653 **APPLICATION USAGE**

10654 None.

10655 **RATIONALE**

10656 None.

10657 **FUTURE DIRECTIONS**

10658 None.

10659 **SEE ALSO**

10660 **<fcntl.h>**, **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *sem_destroy()*,
10661 *sem_getvalue()*, *sem_init()*, *sem_open()*, *sem_post()*, *sem_timedwait()*, *sem_trywait()*, *sem_unlink()*,
10662 *sem_wait()*

10663 **CHANGE HISTORY**

10664 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

10665 **Issue 6**10666 The **<semaphore.h>** header is marked as part of the Semaphores option.

10667 The Open Group Corrigendum U021/3 is applied, adding a description of SEM_FAILED.

10668 The *sem_timedwait()* function is added for alignment with IEEE Std 1003.1d-1999.10669 The **restrict** keyword is added to the prototypes for *sem_getvalue()* and *sem_timedwait()*.

10670 **NAME**

10671 setjmp.h — stack environment declarations

10672 **SYNOPSIS**

10673 #include <setjmp.h>

10674 **DESCRIPTION**

10675 CX Some of the functionality described on this reference page extends the ISO C standard.
 10676 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 10677 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these
 10678 symbols in this header.

10679 CX The <setjmp.h> header shall define the array types **jmp_buf** and **sigjmp_buf**.

10680 The following shall be declared as functions and may also be defined as macros. Function |
 10681 prototypes shall be provided. |

```
10682 void longjmp(jmp_buf, int);
10683 CX void siglongjmp(sigjmp_buf, int);
10684 XSI void _longjmp(jmp_buf, int);
10685
```

10686 The following may be declared as a function, or defined as a macro, or both. Function prototypes |
 10687 shall be provided. |

```
10688 int setjmp(jmp_buf);
10689 CX int sigsetjmp(sigjmp_buf, int);
10690 XSI int _setjmp(jmp_buf);
10691
```

10692 **APPLICATION USAGE**

10693 None.

10694 **RATIONALE**

10695 None.

10696 **FUTURE DIRECTIONS**

10697 None.

10698 **SEE ALSO**

10699 The System Interfaces volume of IEEE Std 1003.1-200x, *longjmp()*, *_longjmp()*, *setjmp()*,
 10700 *siglongjmp()*, *sigsetjmp()*

10701 **CHANGE HISTORY**

10702 First released in Issue 1.

10703 **Issue 6**

10704 Extensions beyond the ISO C standard are now marked.

10705 NAME

10706 signal.h — signals

10707 SYNOPSIS

10708 #include <signal.h>

10709 DESCRIPTION

10710 CX Some of the functionality described on this reference page extends the ISO C standard.
10711 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
10712 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these
10713 symbols in this header.

10714 The <signal.h> header shall define the following symbolic constants, each of which expands to a
10715 distinct constant expression of the type:

10716 void (*)(int)

10717 whose value matches no declarable function.

10718 SIG_DFL Request for default signal handling.

10719 SIG_ERR Return value from *signal()* in case of error.

10720 CX SIG_HOLD Request that signal be held.

10721 SIG_IGN Request that signal be ignored.

10722 The following data types shall be defined through **typedef**:

10723 **sig_atomic_t** Possibly volatile-qualified integer type of an object that can be accessed as
10724 an atomic entity, even in the presence of asynchronous interrupts.

10725 CX **sigset_t** Integer or structure type of an object used to represent sets of signals.

10726 CX **pid_t** As described in <sys/types.h>.

10727 RTS The <signal.h> header shall define the **sigevent** structure, which has at least the following
10728 members:

10729	int	sigev_notify	Notification type.
10730	int	sigev_signo	Signal number.
10731	union sigval	sigev_value	Signal value.
10732	void (*)(union sigval)	sigev_notify_function	Notification function.
10733	(pthread_attr_t *)	sigev_notify_attributes	Notification attributes.

10734 The following values of *sigev_notify* shall be defined:

10735 SIGEV_NONE No asynchronous notification is delivered when the event of interest
10736 occurs.

10737 SIGEV_SIGNAL A queued signal, with an application-defined value, is generated when
10738 the event of interest occurs.

10739 SIGEV_THREAD A notification function is called to perform notification.

10740 The **sigval** union shall be defined as:

10741	int	sival_int	Integer signal value.
10742	void *	sival_ptr	Pointer signal value.

10743 This header shall also declare the macros SIGRTMIN and SIGRTMAX, which evaluate to integer
10744 expressions, and specify a range of signal numbers that are reserved for application use and for
10745 which the realtime signal behavior specified in this volume of IEEE Std 1003.1-200x is supported.

10746 The signal numbers in this range do not overlap any of the signals specified in the following
 10747 table.

10748 The range SIGRTMIN through SIGRTMAX inclusive shall include at least {RTSIG_MAX} signal
 10749 numbers.

10750 It is implementation-defined whether realtime signal behavior is supported for other signals.

10751 This header also declares the constants that are used to refer to the signals that occur in the
 10752 system. Signals defined here begin with the letters SIG. Each of the signals have distinct positive
 10753 integer values. The value 0 is reserved for use as the null signal (see *kill()*). Additional
 10754 implementation-defined signals may occur in the system.

10755 cx The ISO C standard only requires the signal names SIGABRT, SIGFPE, SIGILL, SIGINT,
 10756 SIGSEGV, and SIGTERM to be defined.

10757 The following signals shall be supported on all implementations (default actions are explained
 10758 below the table):

10759

10760

Signal	Default Action	Description
SIGABRT	A	Process abort signal.
SIGALRM	T	Alarm clock.
SIGBUS	A	Access to an undefined portion of a memory object.
SIGCHLD	I	Child process terminated, stopped, or continued.
SIGCONT	C	Continue executing, if stopped.
SIGFPE	A	Erroneous arithmetic operation.
SIGHUP	T	Hangup.
SIGILL	A	Illegal instruction.
SIGINT	T	Terminal interrupt signal.
SIGKILL	T	Kill (cannot be caught or ignored).
SIGPIPE	T	Write on a pipe with no one to read it.
SIGQUIT	A	Terminal quit signal.
SIGSEGV	A	Invalid memory reference.
SIGSTOP	S	Stop executing (cannot be caught or ignored).
SIGTERM	T	Termination signal.
SIGTSTP	S	Terminal stop signal.
SIGTTIN	S	Background process attempting read.
SIGTTOU	S	Background process attempting write.
SIGUSR1	T	User-defined signal 1.
SIGUSR2	T	User-defined signal 2.
SIGPOLL	T	Pollable event.
SIGPROF	T	Profiling timer expired.
SIGSYS	A	Bad system call.
SIGTRAP	A	Trace/breakpoint trap.
SIGURG	I	High bandwidth data is available at a socket.
SIGVTALRM	T	Virtual timer expired.
SIGXCPU	A	CPU time limit exceeded.
SIGXFSZ	A	File size limit exceeded.

10790 The default actions are as follows:

10791 T Abnormal termination of the process. The process is terminated with all the consequences
 10792 of *_exit()* except that the status made available to *wait()* and *waitpid()* indicates abnormal
 10793 termination by the specified signal.

10794 A Abnormal termination of the process. |
10795 XSI Additionally, implementation-defined abnormal termination actions, such as creation of a |
10796 core file, may occur. |
10797 I Ignore the signal.
10798 S Stop the process.
10799 C Continue the process, if it is stopped; otherwise, ignore the signal.

10800 CX The header shall provide a declaration of **struct sigaction**, including at least the following
10801 members:

```

10802 void (*sa_handler)(int) What to do on receipt of signal.
10803 sigset_t sa_mask Set of signals to be blocked during execution
10804 of the signal handling function.
10805 int sa_flags Special flags.
10806 void (*)(int, siginfo_t *, void *) sa_sigaction
10807 Pointer to signal handler function or one
10808 of the macros SIG_IGN or SIG_DFL.

```

10809

10810 XSI The storage occupied by *sa_handler* and *sa_sigaction* may overlap, and a portable program must
10811 not use both simultaneously.

10812 The following shall be declared as constants:

10813 CX SA_NOCLDSTOP Do not generate SIGCHLD when children stop |
10814 XSI or stopped children continue. |

10815 CX SIG_BLOCK The resulting set is the union of the current set and the signal set pointed
10816 to by the argument *set*.

10817 CX SIG_UNBLOCK The resulting set is the intersection of the current set and the complement
10818 of the signal set pointed to by the argument *set*.

10819 CX SIG_SETMASK The resulting set is the signal set pointed to by the argument *set*.

10820 XSI SA_ONSTACK Causes signal delivery to occur on an alternate stack.

10821 XSI SA_RESETHAND Causes signal dispositions to be set to SIG_DFL on entry to signal
10822 handlers.

10823 XSI SA_RESTART Causes certain functions to become restartable.

10824 XSI SA_SIGINFO Causes extra information to be passed to signal handlers at the time of
10825 receipt of a signal.

10826 XSI SA_NOCLDWAIT Causes implementations not to create zombie processes on child death.

10827 XSI SA_NODEFER Causes signal not to be automatically blocked on entry to signal handler.

10828 XSI SS_ONSTACK Process is executing on an alternate signal stack.

10829 XSI SS_DISABLE Alternate signal stack is disabled.

10830 XSI MINSIGSTKSZ Minimum stack size for a signal handler.

10831 XSI SIGSTKSZ Default size in bytes for the alternate signal stack.

10832 XSI The **ucontext_t** structure shall be defined through **typedef** as described in <ucontext.h>.

10833 The **mcontext_t** type shall be defined through **typedef** as described in <ucontext.h>.

10834 The <signal.h> header shall define the **stack_t** type as a structure that includes at least the
 10835 following members:

10836	void	*ss_sp	Stack base or pointer.
10837	size_t	ss_size	Stack size.
10838	int	ss_flags	Flags.

10839 The <signal.h> header shall define the **sigstack** structure that includes at least the following
 10840 members:

10841	int	ss_onstack	Non-zero when signal stack is in use.
10842	void	*ss_sp	Signal stack pointer.

10843

10844 CX The <signal.h> header shall define the **siginfo_t** type as a structure that includes at least the
 10845 following members:

10846 CX	int	si_signo	Signal number.
10847 XSI	int	si_errno	If non-zero, an <i>errno</i> value associated with 10848 this signal, as defined in <errno.h>.
10849 CX	int	si_code	Signal code.
10850 XSI	pid_t	si_pid	Sending process ID.
10851	uid_t	si_uid	Real user ID of sending process.
10852	void	*si_addr	Address of faulting instruction.
10853	int	si_status	Exit value or signal.
10854	long	si_band	Band event for SIGPOLL.
10855 RTS	union signal	si_value	Signal value.
10856			

10857 The macros specified in the **Code** column of the following table are defined for use as values of
 10858 XSI *si_code* that are signal-specific or non-signal-specific reasons why the signal was generated.

10859

10860

10861 XSI

10862

10863

10864

10865

10866

10867

10868

10869

10870

10871

10872

10873

10874

10875

10876

10877

10878

10879

10880

10881

10882

10883

10884

10885

10886

10887

10888

10889

10890

10891

10892

10893

10894

10895

10896 CX

10897

10898

10899

10900

10901

10902

Signal	Code	Reason
SIGILL	ILL_ILLOPC	Illegal opcode.
	ILL_ILLOPN	Illegal operand.
	ILL_ILLADR	Illegal addressing mode.
	ILL_ILLTRP	Illegal trap.
	ILL_PRVOPC	Privileged opcode.
	ILL_PRVREG	Privileged register.
	ILL_COPROC	Coprocessor error.
	ILL_BADSTK	Internal stack error.
SIGFPE	FPE_INTDIV	Integer divide by zero.
	FPE_INTOVF	Integer overflow.
	FPE_FLTDIV	Floating-point divide by zero.
	FPE_FLTOVF	Floating-point overflow.
	FPE_FLTUND	Floating-point underflow.
	FPE_FLTRES	Floating-point inexact result.
	FPE_FLTINV	Invalid floating-point operation.
	FPE_FLTSUB	Subscript out of range.
SIGSEGV	SEGV_MAPERR	Address not mapped to object.
	SEGV_ACCERR	Invalid permissions for mapped object.
SIGBUS	BUS_ADRALN	Invalid address alignment.
	BUS_ADRERR	Non-existent physical address.
	BUS_OBJERR	Object specific hardware error.
SIGTRAP	TRAP_BRKPT	Process breakpoint.
	TRAP_TRACE	Process trace trap.
SIGCHLD	CLD_EXITED	Child has exited.
	CLD_KILLED	Child has terminated abnormally and did not create a core file.
	CLD_DUMPED	Child has terminated abnormally and created a core file.
	CLD_TRAPPED	Traced child has trapped.
	CLD_STOPPED	Child has stopped.
	CLD_CONTINUED	Stopped child has continued.
SIGPOLL	POLL_IN	Data input available.
	POLL_OUT	Output buffers available.
	POLL_MSG	Input message available.
	POLL_ERR	I/O error.
	POLL_PRI	High priority input available.
	POLL_HUP	Device disconnected.
Any	SI_USER	Signal sent by <i>kill()</i> .
	SI_QUEUE	Signal sent by the <i>sigqueue()</i> .
	SI_TIMER	Signal generated by expiration of a timer set by <i>timer_settime()</i> .
	SI_ASYNCIO	Signal generated by completion of an asynchronous I/O request.
	SI_MESGQ	Signal generated by arrival of a message on an empty message queue.

10903 XSI

10904

10905

10906

10907

Implementations may support additional *si_code* values not included in this list, may generate values included in this list under circumstances other than those described in this list, and may contain extensions or limitations that prevent some values from being generated. Implementations do not generate a different value from the ones described in this list for circumstances described in this list.

10908 In addition, the following signal-specific information shall be available:

10909

10910

10911

10912

10913

10914

10915

10916

10917

10918

Signal	Member	Value
SIGILL SIGFPE	void * si_addr	Address of faulting instruction.
SIGSEGV SIGBUS	void * si_addr	Address of faulting memory reference.
SIGCHLD	pid_t si_pid int si_status uid_t si_uid	Child process ID. Exit value or signal. Real user ID of the process that sent the signal.
SIGPOLL	long si_band	Band event for POLL_IN, POLL_OUT, or POLL_MSG.

10919

For some implementations, the value of *si_addr* may be inaccurate.

10920

The following shall be declared as functions and may also be defined as macros:

10921 XSI

10922 CX

10923 XSI

10924 THR

10925

10926

10927 CX

10928

10929

10930 XSI

10931 CX

10932

10933

10934 XSI

10935

10936

10937 CX

10938

10939 XSI

10940 CX

10941

10942 RTS

10943 XSI

10944

10945 CX

10946 RTS

10947

10948 CX

10949 RTS

10950

```

void (*bsd_signal(int, void (*)(int)))(int);
int kill(pid_t, int);
int killpg(pid_t, int);
int pthread_kill(pthread_t, int);
int pthread_sigmask(int, const sigset_t *, sigset_t *);
int raise(int);
int sigaction(int, const struct sigaction *restrict,
              struct sigaction *restrict);
int sigaddset(sigset_t *, int);
int sigaltstack(const stack_t *restrict, stack_t *restrict);
int sigdelset(sigset_t *, int);
int sigemptyset(sigset_t *);
int sigfillset(sigset_t *);
int sighold(int);
int sigignore(int);
int siginterrupt(int, int);
int sigismember(const sigset_t *, int);
void (*signal(int, void (*)(int)))(int);
int sigpause(int);
int sigpending(sigset_t *);
int sigprocmask(int, const sigset_t *restrict, sigset_t *restrict);
int sigqueue(pid_t, int, const union sigval);
int sigrelse(int);
void (*sigset(int, void (*)(int)))(int);
int sigsuspend(const sigset_t *);
int sigtimedwait(const sigset_t *restrict, siginfo_t *restrict,
                const struct timespec *restrict);
int sigwait(const sigset_t *restrict, int *restrict);
int sigwaitinfo(const sigset_t *restrict, siginfo_t *restrict);

```

- 10951 CX Inclusion of the **<signal.h>** header may make visible all symbols from the **<time.h>** header.
- 10952 **APPLICATION USAGE**
- 10953 None.
- 10954 **RATIONALE**
- 10955 None.
- 10956 **FUTURE DIRECTIONS**
- 10957 None.
- 10958 **SEE ALSO**
- 10959 **<errno.h>**, **<stropts.h>**, **<sys/types.h>**, **<time.h>**, **<ucontext.h>**, the System Interfaces volume of
10960 IEEE Std 1003.1-200x, *alarm()*, *bsd_signal()*, *ioctl()*, *kill()*, *killpg()*, *raise()*, *sigaction()*, *sigaddset()*,
10961 *sigaltstack()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *siginterrupt()*, *sigismember()*, *signal()*,
10962 *sigpending()*, *sigprocmask()*, *sigqueue()*, *sigsuspend()*, *sigwaitinfo()*, *wait()*, *waitid()*
- 10963 **CHANGE HISTORY**
- 10964 First released in Issue 1.
- 10965 **Issue 5**
- 10966 The DESCRIPTION is updated for alignment with POSIX Realtime Extension and the POSIX
10967 Threads Extension.
- 10968 The default action for SIGURG is changed for i to iii. The function prototype for *sigmask()* is
10969 removed.
- 10970 **Issue 6**
- 10971 The Open Group Corrigendum U035/2 is applied. In the DESCRIPTION, the wording for
10972 abnormal termination is clarified.
- 10973 The Open Group Corrigendum U028/8 is applied, correcting the prototype for the *sigset()*
10974 function.
- 10975 The Open Group Corrigendum U026/3 is applied, correcting the type of the *sigev_notify_function*
10976 function member of the **sigevent** structure.
- 10977 The following new requirements on POSIX implementations derive from alignment with the
10978 Single UNIX Specification:
- 10979 • The SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, and SIGTTOU signals are now
10980 mandated. This is also a FIPS requirement.
 - 10981 • The **pid_t** definition is mandated.
- 10982 The RT markings are now changed to RTS to denote that the semantics are part of the Realtime
10983 Signals Extension option.
- 10984 The **restrict** keyword is added to the prototypes for *sigaction()*, *sigaltstack()*, *sigprocmask()*,
10985 *sigtimedwait()*, *sigwait()*, and *sigwaitinfo()*.
- 10986 IEEE PASC Interpretation 1003.1 #85 is applied, adding the statement that symbols from
10987 **<time.h>** may be made visible when **<signal.h>** is included. Extensions beyond the ISO C
10988 standard are now marked.

10989 **NAME**

10990 spawn.h — spawn (ADVANCED REALTIME)

10991 **SYNOPSIS**

10992 SPN #include <spawn.h>

10993

10994 **DESCRIPTION**

10995 The <spawn.h> header shall define the **posix_spawnattr_t** and **posix_spawn_file_actions_t**
 10996 types used in performing spawn operations.

10997 The <spawn.h> header shall define the flags that may be set in a **posix_spawnattr_t** object using
 10998 the *posix_spawnattr_setflags()* function:

- 10999 POSIX_SPAWN_RESETEIDS
- 11000 POSIX_SPAWN_SETPGROUP
- 11001 PS POSIX_SPAWN_SETSCHEDPARAM
- 11002 POSIX_SPAWN_SETSCHEDULER
- 11003 POSIX_SPAWN_SETSIGDEF
- 11004 POSIX_SPAWN_SETSIGMASK

11005 The following shall be declared as functions and may also be defined as macros. Function |
 11006 prototypes shall be provided. |

```

11007 int    posix_spawn(pid_t *restrict, const char *restrict,
11008                const posix_spawn_file_actions_t *,
11009                const posix_spawnattr_t *restrict, char *const [restrict],
11010                char *const [restrict]);
11011 int    posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *,
11012                int);
11013 int    posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *,
11014                int, int);
11015 int    posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *restrict,
11016                int, const char *restrict, int, mode_t);
11017 int    posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *);
11018 int    posix_spawn_file_actions_init(posix_spawn_file_actions_t *);
11019 int    posix_spawnattr_destroy(posix_spawnattr_t *);
11020 int    posix_spawnattr_getsigdefault(const posix_spawnattr_t *restrict,
11021                sigset_t *restrict);
11022 int    posix_spawnattr_getflags(const posix_spawnattr_t *restrict,
11023                short *restrict);
11024 int    posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict,
11025                pid_t *restrict);
11026 PS int    posix_spawnattr_getschedparam(const posix_spawnattr_t *restrict,
11027                struct sched_param *restrict);
11028 int    posix_spawnattr_getschedpolicy(const posix_spawnattr_t *restrict,
11029                int *restrict);
11030 int    posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict,
11031                sigset_t *restrict);
11032 int    posix_spawnattr_init(posix_spawnattr_t *);
11033 int    posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict,
11034                const sigset_t *restrict);
11035 int    posix_spawnattr_setflags(posix_spawnattr_t *, short);
11036 int    posix_spawnattr_setpgroup(posix_spawnattr_t *, pid_t);
    
```

11037 PS

```
11038 int    posix_spawnattr_setschedparam(posix_spawnattr_t *restrict,  
11039         const struct sched_param *restrict);  
11040 int    posix_spawnattr_setschedpolicy(posix_spawnattr_t *, int);  
11041 int    posix_spawnattr_setsigmask(posix_spawnattr_t *restrict,  
11042         const sigset_t *restrict);  
11043 int    posix_spawnnp(pid_t *restrict, const char *restrict,  
11044         const posix_spawn_file_actions_t *,  
11045         const posix_spawnattr_t *restrict,  
11046         char *const [restrict], char *const [restrict]);
```

11047 Inclusion of the **<spawn.h>** header may make visible symbols defined in the **<sched.h>**,
11048 **<signal.h>**, and **<sys/types.h>** headers.

11049 APPLICATION USAGE

11050 None.

11051 RATIONALE

11052 None.

11053 FUTURE DIRECTIONS

11054 None.

11055 SEE ALSO

11056 **<sched.h>**, **<semaphore.h>**, **<signal.h>**, **<sys/types.h>**, the System Interfaces volume of
11057 IEEE Std 1003.1-200x, *posix_spawnattr_destroy()*, *posix_spawnattr_getsigdefault()*,
11058 *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*,
11059 *posix_spawnattr_getschedpolicy()*, *posix_spawnattr_getsigmask()*, *posix_spawnattr_init()*,
11060 *posix_spawnattr_setsigdefault()*, *posix_spawnattr_setflags()*, *posix_spawnattr_setpgroup()*,
11061 *posix_spawnattr_setschedparam()*, *posix_spawnattr_setschedpolicy()*, *posix_spawnattr_setsigmask()*,
11062 *posix_spawn()*, *posix_spawn_file_actions_addclose()*, *posix_spawn_file_actions_adddup2()*,
11063 *posix_spawn_file_actions_addopen()*, *posix_spawn_file_actions_destroy()*,
11064 *posix_spawn_file_actions_init()*, *posix_spawnnp()*

11065 CHANGE HISTORY

11066 First released in Issue 6. Included for alignment with IEEE Std 1003.1d-1999.

11067 The **restrict** keyword is added to the prototypes for *posix_spawn()*,
11068 *posix_spawn_file_actions_addopen()*, *posix_spawnattr_getsigdefault()*, *posix_spawnattr_getflags()*,
11069 *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*, *posix_spawnattr_getschedpolicy()*,
11070 *posix_spawnattr_getsigmask()*, *posix_spawnattr_setsigdefault()*, *posix_spawnattr_setschedparam()*,
11071 *posix_spawnattr_setsigmask()*, and *posix_spawnnp()*.

11072 **NAME**11073 **stdarg.h** — handle variable argument list11074 **SYNOPSIS**

11075 #include <stdarg.h>

11076 void va_start(va_list *ap*, *argN*);11077 void va_copy(va_list *dest*, va_list *src*);11078 type va_arg(va_list *ap*, *type*);11079 void va_end(va_list *ap*);11080 **DESCRIPTION**

11081 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 11082 conflict between the requirements described here and the ISO C standard is unintentional. This
 11083 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11084 The <stdarg.h> header shall contain a set of macros which allows portable functions that accept
 11085 variable argument lists to be written. Functions that have variable argument lists (such as
 11086 *printf()*) but do not use these macros, are inherently non-portable, as different systems use
 11087 different argument-passing conventions.

11088 The type **va_list** shall be defined for variables used to traverse the list.

11089 The *va_start()* macro is invoked to initialize *ap* to the beginning of the list before any calls to
 11090 *va_arg()*.

11091 The *va_copy()* macro initializes as a copy of *src*, as if the *va_start()* macro had been applied to
 11092 *dest* followed by the same sequence of uses of the *va_arg()* macro as had previously been used to
 11093 reach the present state of *src*. Neither the *va_copy()* nor *va_start()* macro shall be invoked to
 11094 reinitialize *dest* without an intervening invocation of the *va_end()* macro for the same *dest*.

11095 The object *ap* may be passed as an argument to another function; if that function invokes the
 11096 *va_arg()* macro with parameter *ap*, the value of *ap* in the calling function is unspecified and shall
 11097 be passed to the *va_end()* macro prior to any further reference to *ap*. The parameter *argN* is the
 11098 identifier of the rightmost parameter in the variable parameter list in the function definition (the
 11099 one just before the ...). If the parameter *argN* is declared with the **register** storage class, with a
 11100 function type or array type, or with a type that is not compatible with the type that results after
 11101 application of the default argument promotions, the behavior is undefined.

11102 The *va_arg()* macro shall return the next argument in the list pointed to by *ap*. Each invocation
 11103 of *va_arg()* modifies *ap* so that the values of successive arguments are returned in turn. The *type*
 11104 parameter is the type the argument is expected to be. This is the type name specified such that
 11105 the type of a pointer to an object that has the specified type can be obtained simply by suffixing
 11106 a '*' to *type*. Different types can be mixed, but it is up to the routine to know what type of
 11107 argument is expected.

11108 The *va_end()* macro is used to clean up; it invalidates *ap* for use (unless *va_start()* or *va_copy()* is
 11109 invoked again).

11110 Each invocation of the *va_start()* and *va_copy()* macros shall be matched by a corresponding
 11111 invocation of the *va_end()* macro in the same function.

11112 Multiple traversals, each bracketed by *va_start()* ... *va_end()*, are possible.

11113 **EXAMPLES**

11114 This example is a possible implementation of *execl()*:

11115 #include <stdarg.h>

```
11116     #define  MAXARGS      31
11117     /*
11118     * execl is called by
11119     * execl(file, arg1, arg2, ..., (char *) (0));
11120     */
11121     int execl(const char *file, const char *args, ...)
11122     {
11123         va_list ap;
11124         char *array[MAXARGS];
11125         int argno = 0;
11126         va_start(ap, args);
11127         while (args != 0) {
11128             array[argno++] = args;
11129             args = va_arg(ap, const char *);
11130         }
11131         va_end(ap);
11132         return execv(file, array);
11133     }
```

11134 APPLICATION USAGE

11135 It is up to the calling routine to communicate to the called routine how many arguments there
11136 are, since it is not always possible for the called routine to determine this in any other way. For
11137 example, *execl()* is passed a null pointer to signal the end of the list. The *printf()* function can tell
11138 how many arguments are there by the *format* argument.

11139 RATIONALE

11140 None.

11141 FUTURE DIRECTIONS

11142 None.

11143 SEE ALSO

11144 The System Interfaces volume of IEEE Std 1003.1-200x, *exec()*, *printf()*

11145 CHANGE HISTORY

11146 First released in Issue 4. Derived from the ANSI C standard.

11147 **NAME**

11148 stdbool.h — boolean type and values

11149 **SYNOPSIS**

11150 #include <stdbool.h>

11151 **DESCRIPTION**

11152 cx The functionality described on this reference page is aligned with the ISO C standard. Any
11153 conflict between the requirements described here and the ISO C standard is unintentional. This
11154 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11155 The <stdbool.h> header shall define the following macros:

11156 bool Expands to **_Bool**.

11157 true Expands to the integer constant 1.

11158 false Expands to the integer constant 0.

11159 __bool_true_false_are_defined

11160 Expands to the integer constant 1.

11161 An application may undefine and then possibly redefine the macros bool, true, and false.

11162 **APPLICATION USAGE**

11163 None.

11164 **RATIONALE**

11165 None.

11166 **FUTURE DIRECTIONS**

11167 The ability to undefine and redefine the macros bool, true, and false is an obsolescent feature
11168 and may be withdrawn in the future.

11169 **SEE ALSO**

11170 None.

11171 **CHANGE HISTORY**

11172 First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

11173 **NAME**11174 **stddef.h** — standard type definitions11175 **SYNOPSIS**

11176 #include <stddef.h>

11177 **DESCRIPTION**

11178 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
11179 conflict between the requirements described here and the ISO C standard is unintentional. This
11180 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

11181 The <**stddef.h**> header shall define the following macros:11182 **NULL** Null pointer constant.11183 **offsetof**(*type*, *member-designator*)

11184 Integer constant expression of type **size_t**, the value of which is the offset in bytes
11185 to the structure member (*member-designator*), from the beginning of its structure
11186 (*type*).

11187 The <**stddef.h**> header shall define the following types:11188 **ptrdiff_t** Signed integer type of the result of subtracting two pointers.

11189 **wchar_t** Integer type whose range of values can represent distinct wide-character codes for
11190 all members of the largest character set specified among the locales supported by
11191 the compilation environment: the null character has the code value 0 and each
11192 member of the portable character set has a code value equal to its value when used
11193 as the lone character in an integer character constant. |

11194 **size_t** Unsigned integer type of the result of the *sizeof* operator.

11195 The implementation shall support one or more programming environments in which the widths |
11196 of **ptrdiff_t**, **size_t**, and **wchar_t** are no greater than the width of type **long**. The names of these |
11197 programming environments can be obtained using the *confstr()* function or the *getconf* utility. |

11198 **APPLICATION USAGE**

11199 None.

11200 **RATIONALE**

11201 None.

11202 **FUTURE DIRECTIONS**

11203 None.

11204 **SEE ALSO**

11205 <**wchar.h**>, <**sys/types.h**>, the System Interfaces volume of IEEE Std 1003.1-200x, *confstr()*, the |
11206 Shell and Utilities volume of IEEE Std 1003.1-200x, *getconf* |

11207 **CHANGE HISTORY**

11208 First released in Issue 4. Derived from the ANSI C standard.

11209 **NAME**

11210 stdint.h — integer types

11211 **SYNOPSIS**

11212 #include <stdint.h>

11213 **DESCRIPTION**

11214 **CX** Some of the functionality described on this reference page extends the ISO C standard.
 11215 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 11216 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these
 11217 symbols in this header.

11218 The <stdint.h> header shall declare sets of integer types having specified widths, and shall
 11219 define corresponding sets of macros. It shall also define macros that specify limits of integer
 11220 types corresponding to types defined in other standard headers.

11221 **Note:** The “width” of an integer type is the number of bits used to store its value in a pure binary
 11222 system; the actual type may use more bits than that (for example, a 28-bit type could be stored
 11223 in 32 bits of actual storage). An N -bit signed type has values in the range -2^{N-1} or $1-2^{N-1}$ to
 11224 $2^{N-1}-1$, while an N -bit unsigned type has values in the range 0 to 2^N-1 .

11225 Types are defined in the following categories:

- 11226 • Integer types having certain exact widths
- 11227 • Integer types having at least certain specified widths
- 11228 • Fastest integer types having at least certain specified widths
- 11229 • Integer types wide enough to hold pointers to objects
- 11230 • Integer types having greatest width

11231 (Some of these types may denote the same type.)

11232 Corresponding macros specify limits of the declared types and construct suitable constants.

11233 For each type described herein that the implementation provides, the <stdint.h> header shall
 11234 declare that **typedef** name and define the associated macros. Conversely, for each type described
 11235 herein that the implementation does not provide, the <stdint.h> header shall not declare that
 11236 **typedef** name, nor shall it define the associated macros. An implementation shall provide those
 11237 types described as required, but need not provide any of the others (described as optional).

11238 **Integer Types**

11239 When **typedef** names differing only in the absence or presence of the initial u are defined, they
 11240 shall denote corresponding signed and unsigned types as described in the ISO/IEC 9899:1999
 11241 standard, Section 6.2.5; an implementation providing one of these corresponding types shall also
 11242 provide the other.

11243 In the following descriptions, the symbol N represents an unsigned decimal integer with no
 11244 leading zeros (for example, 8 or 24, but not 04 or 048).

- 11245 • Exact-width integer types

11246 The **typedef** name **int N _t** designates a signed integer type with width N , no padding bits,
 11247 and a two’s-complement representation. Thus, **int8_t** denotes a signed integer type with a
 11248 width of exactly 8 bits.

11249 The **typedef** name **uint N _t** designates an unsigned integer type with width N . Thus,
 11250 **uint24_t** denotes an unsigned integer type with a width of exactly 24 bits.

11251 cx The following types are required:

11252 **int8_t**

11253 **int16_t**

11254 **int32_t**

11255 **uint8_t**

11256 **uint16_t**

11257 **uint32_t**

11258

11259 If an implementation provides integer types with width 64 that meet these requirements,
11260 then the following types are required:

11261 **int64_t**

11262 **uint64_t**

11263 cx In particular, this will be the case if any of the following are true:

11264 — The implementation supports the `_POSIX_V6_ILP32_OFFBIG` programming
11265 environment and the application is being built in the `_POSIX_V6_ILP32_OFFBIG`
11266 programming environment (see the Shell and Utilities volume of IEEE Std 1003.1-200x,
11267 *c99*, Programming Environments).

11268 — The implementation supports the `_POSIX_V6_LP64_OFF64` programming environment
11269 and the application is being built in the `_POSIX_V6_LP64_OFF64` programming
11270 environment.

11271 — The implementation supports the `_POSIX_V6_LPBIG_OFFBIG` programming
11272 environment and the application is being built in the `_POSIX_V6_LPBIG_OFFBIG`
11273 programming environment.

11274 All other types are of this form optional.

11275 • Minimum-width integer types

11276 The **typedef** name **int_leastN_t** designates a signed integer type with a width of at least *N*,
11277 such that no signed integer type with lesser size has at least the specified width. Thus,
11278 **int_least32_t** denotes a signed integer type with a width of at least 32 bits.

11279 The **typedef** name **uint_leastN_t** designates an unsigned integer type with a width of at least
11280 *N*, such that no unsigned integer type with lesser size has at least the specified width. Thus,
11281 **uint_least16_t** denotes an unsigned integer type with a width of at least 16 bits.

11282 The following types are required:

11283 **int_least8_t**

11284 **int_least16_t**

11285 **int_least32_t**

11286 **int_least64_t**

11287 **uint_least8_t**

11288 **uint_least16_t**

11289 **uint_least32_t**

11290 **uint_least64_t**

11291 All other types of this form are optional.

11292 • Fastest minimum-width integer types

11293 Each of the following types designates an integer type that is usually fastest to operate with
11294 among all integer types that have at least the specified width.

11295 The designated type is not guaranteed to be fastest for all purposes; if the implementation
 11296 has no clear grounds for choosing one type over another, it will simply pick some integer
 11297 type satisfying the signedness and width requirements.

11298 The **typedef** name **int_fastN_t** designates the fastest signed integer type with a width of at
 11299 least *N*. The **typedef** name **uint_fastN_t** designates the fastest unsigned integer type with a
 11300 width of at least *N*.

11301 The following types are required:

11302 **int_fast8_t**
 11303 **int_fast16_t**
 11304 **int_fast32_t**
 11305 **int_fast64_t**
 11306 **uint_fast8_t**
 11307 **uint_fast16_t**
 11308 **uint_fast32_t**
 11309 **uint_fast64_t**

11310 All other types of this form are optional.

11311 • Integer types capable of holding object pointers

11312 The following type designates a signed integer type with the property that any valid pointer
 11313 to **void** can be converted to this type, then converted back to a pointer to **void**, and the result
 11314 will compare equal to the original pointer:

11315 **intptr_t**

11316 The following type designates an unsigned integer type with the property that any valid
 11317 pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and
 11318 the result will compare equal to the original pointer:

11319 **uintptr_t**

11320 XSI On XSI-conformant systems, the **intptr_t** and **uintptr_t** types are required; otherwise, they are
 11321 optional.

11322 • Greatest-width integer types

11323 The following type designates a signed integer type capable of representing any value of any
 11324 signed integer type:

11325 **intmax_t**

11326 The following type designates an unsigned integer type capable of representing any value of
 11327 any unsigned integer type:

11328 **uintmax_t**

11329 These types are required.

11330 **Note:** Applications can test for optional types by using the corresponding limit macro from **Limits of**
 11331 **Specified-Width Integer Types** (on page 316).

11332 **Limits of Specified-Width Integer Types**

11333 The following macros specify the minimum and maximum limits of the types declared in the
 11334 **<stdint.h>** header. Each macro name corresponds to a similar type name in **Integer Types** (on
 11335 page 313).

11336 Each instance of any defined macro shall be replaced by a constant expression suitable for use in
 11337 **#if** preprocessing directives, and this expression shall have the same type as would an
 11338 expression that is an object of the corresponding type converted according to the integer
 11339 promotions. Its implementation-defined value shall be equal to or greater in magnitude
 11340 (absolute value) than the corresponding value given below, with the same sign, except where
 11341 stated to be exactly the given value.

11342 • Limits of exact-width integer types

11343 — Minimum values of exact-width signed integer types:

11344 {INTN_MIN} Exactly $-(2^{N-1})$

11345 — Maximum values of exact-width signed integer types:

11346 {INTN_MAX} Exactly $2^{N-1} - 1$

11347 — Maximum values of exact-width unsigned integer types:

11348 {UINTN_MAX} Exactly $2^N - 1$

11349 • Limits of minimum-width integer types

11350 — Minimum values of minimum-width signed integer types:

11351 {INT_LEASTN_MIN} $-(2^{N-1} - 1)$

11352 — Maximum values of minimum-width signed integer types:

11353 {INT_LEASTN_MAX} $2^{N-1} - 1$

11354 — Maximum values of minimum-width unsigned integer types:

11355 {UINT_LEASTN_MAX} $2^N - 1$

11356 • Limits of fastest minimum-width integer types

11357 — Minimum values of fastest minimum-width signed integer types:

11358 {INT_FASTN_MIN} $-(2^{N-1} - 1)$

11359 — Maximum values of fastest minimum-width signed integer types:

11360 {INT_FASTN_MAX} $2^{N-1} - 1$

11361 — Maximum values of fastest minimum-width unsigned integer types:

11362 {UINT_FASTN_MAX} $2^N - 1$

11363 • Limits of integer types capable of holding object pointers

11364 — Minimum value of pointer-holding signed integer type:

11365 {INTPTR_MIN} $-(2^{15} - 1)$

11366 — Maximum value of pointer-holding signed integer type:

11367 {INTPTR_MAX} $2^{15} - 1$

11368 — Maximum value of pointer-holding unsigned integer type:

- 11369 {UINTPTR_MAX} $2^{16} - 1$
- 11370 • Limits of greatest-width integer types
- 11371 — Minimum value of greatest-width signed integer type:
- 11372 {INTMAX_MIN} $-(2^{63} - 1)$
- 11373 — Maximum value of greatest-width signed integer type:
- 11374 {INTMAX_MAX} $2^{63} - 1$
- 11375 — Maximum value of greatest-width unsigned integer type:
- 11376 {UINTMAX_MAX} $2^{64} - 1$

11377 **Limits of Other Integer Types**

11378 The following macros specify the minimum and maximum limits of integer types corresponding
 11379 to types defined in other standard headers.

11380 Each instance of these macros shall be replaced by a constant expression suitable for use in #if
 11381 preprocessing directives, and this expression shall have the same type as would an expression
 11382 that is an object of the corresponding type converted according to the integer promotions. Its
 11383 implementation-defined value shall be equal to or greater in magnitude (absolute value) than
 11384 the corresponding value given below, with the same sign.

- 11385 • Limits of **ptrdiff_t**:
- 11386 {PTRDIFF_MIN} -65535
- 11387 {PTRDIFF_MAX} +65535
- 11388 • Limits of **sig_atomic_t**:
- 11389 {SIG_ATOMIC_MIN} See below.
- 11390 {SIG_ATOMIC_MAX} See below.
- 11391 • Limit of **size_t**:
- 11392 {SIZE_MAX} 65535
- 11393 • Limits of **wchar_t**:
- 11394 {WCHAR_MIN} See below.
- 11395 {WCHAR_MAX} See below.
- 11396 • Limits of **wint_t**:
- 11397 {WINT_MIN} See below.
- 11398 {WINT_MAX} See below.

11399 If **sig_atomic_t** (see the <signal.h> header) is defined as a signed integer type, the value of
 11400 {SIG_ATOMIC_MIN} shall be no greater than -127 and the value of {SIG_ATOMIC_MAX} shall
 11401 be no less than 127; otherwise, **sig_atomic_t** shall be defined as an unsigned integer type, and the
 11402 value of {SIG_ATOMIC_MIN} shall be 0 and the value of {SIG_ATOMIC_MAX} shall be no less
 11403 than 255.

11404 If **wchar_t** (see the <stddef.h> header) is defined as a signed integer type, the value of
 11405 {WCHAR_MIN} shall be no greater than -127 and the value of {WCHAR_MAX} shall be no less
 11406 than 127; otherwise, **wchar_t** shall be defined as an unsigned integer type, and the value of
 11407 {WCHAR_MIN} shall be 0 and the value of {WCHAR_MAX} shall be no less than 255.

11408 If **wint_t** (see the <wchar.h> header) is defined as a signed integer type, the value of
 11409 {WINT_MIN} shall be no greater than -32767 and the value of {WINT_MAX} shall be no less
 11410 than 32767; otherwise, **wint_t** shall be defined as an unsigned integer type, and the value of
 11411 {WINT_MIN} shall be 0 and the value of {WINT_MAX} shall be no less than 65535.

11412 **Macros for Integer Constant Expressions**

11413 The following macros expand to integer constant expressions suitable for initializing objects that
 11414 have integer types corresponding to types defined in the <stdint.h> header. Each macro name
 11415 corresponds to a similar type name listed under *Minimum-width integer types* and *Greatest-width*
 11416 *integer types*.

11417 Each invocation of one of these macros shall expand to an integer constant expression suitable
 11418 for use in #if preprocessing directives. The type of the expression shall have the same type as
 11419 would an expression that is an object of the corresponding type converted according to the
 11420 integer promotions. The value of the expression shall be that of the argument.

11421 The argument in any instance of these macros shall be a decimal, octal, or hexadecimal constant
 11422 with a value that does not exceed the limits for the corresponding type.

- 11423 • Macros for minimum-width integer constant expressions

11424 The macro *INTN_C(value)* shall expand to an integer constant expression corresponding to
 11425 the type **int_leastN_t**. The macro *UINTN_C(value)* shall expand to an integer constant
 11426 expression corresponding to the type **uint_leastN_t**. For example, if **uint_least64_t** is a name
 11427 for the type **unsigned long long**, then *UINT64_C(0x123)* might expand to the integer
 11428 constant 0x123ULL.

- 11429 • Macros for greatest-width integer constant expressions

11430 The following macro expands to an integer constant expression having the value specified by
 11431 its argument and the type **intmax_t**:

11432 *INTMAX_C(value)*

11433 The following macro expands to an integer constant expression having the value specified by
 11434 its argument and the type **uintmax_t**:

11435 *UINTMAX_C(value)*

11436 **APPLICATION USAGE**

11437 None.

11438 **RATIONALE**

11439 The <stdint.h> header is a subset of the <inttypes.h> header more suitable for use in
 11440 freestanding environments, which might not support the formatted I/O functions. In some
 11441 environments, if the formatted conversion support is not wanted, using this header instead of
 11442 the <inttypes.h> header avoids defining such a large number of macros.

11443 As a consequence of adding **int8_t** the following are true:

- 11444 • A byte is exactly 8 bits.
- 11445 • {CHAR_BIT} has the value 8, {SCHAR_MAX} has the value 127, {SCHAR_MIN} has the
 11446 value -127 or -128, and {UCHAR_MAX} has the value 255.

11447 **FUTURE DIRECTIONS**

11448 **typedef** names beginning with **int** or **uint** and ending with **_t** may be added to the types defined
 11449 in the <stdint.h> header. Macro names beginning with **INT** or **UINT** and ending with **_MAX**,
 11450 **_MIN**, or **_C** may be added to the macros defined in the <stdint.h> header.

11451 **SEE ALSO**

11452 <signal.h>, <stddef.h>, <wchar.h>, <inttypes.h>

11453 **CHANGE HISTORY**

11454 First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard. |

11455 ISO/IEC 9899: 1999 standard, Technical Corrigendum No. 1 is incorporated. |

11456 **NAME**11457 **stdio.h** — standard buffered input/output11458 **SYNOPSIS**

11459 #include <stdio.h>

11460 **DESCRIPTION**

11461 **CX** Some of the functionality described on this reference page extends the ISO C standard.
 11462 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 11463 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these
 11464 symbols in this header.

11465 The <**stdio.h**> header shall define the following macros as positive integer constant expressions:11466 **BUFSIZ** Size of <**stdio.h**> buffers.11467 **_IOFBF** Input/output fully buffered.11468 **_IOLBF** Input/output line buffered.11469 **_IONBF** Input/output unbuffered.11470 **CX** **L_ctermid** Maximum size of character array to hold *ctermid()* output.11471 **L_tmpnam** Maximum size of character array to hold *tmpnam()* output.11472 **SEEK_CUR** Seek relative to current position.11473 **SEEK_END** Seek relative to end-of-file.11474 **SEEK_SET** Seek relative to start-of-file.11475 The following macros shall be defined as positive integer constant expressions which denote
 11476 implementation limits:11477 {**FILENAME_MAX**} Maximum size in bytes of the longest filename string that the
 11478 implementation guarantees can be opened.11479 {**FOPEN_MAX**} Number of streams which the implementation guarantees can be open
 11480 simultaneously. The value is at least eight.11481 {**TMP_MAX**} Minimum number of unique filenames generated by *tmpnam()*.
 11482 Maximum number of times an application can call *tmpnam()* reliably. The
 11483 **XSI** value of {**TMP_MAX**} is at least 25. On **XSI**-conformant systems, the
 11484 value of {**TMP_MAX**} is at least 10,000.

11485 The following macro name shall be defined as a negative integer constant expression:

11486 **EOF** End-of-file return value.

11487 The following macro name shall be defined as a null pointer constant:

11488 **NULL** Null pointer.

11489 The following macro name shall be defined as a string constant:

11490 **XSI** **P_tmpdir** Default directory prefix for *tmpnam()*.11491 The following shall be defined as expressions of type “pointer to **FILE**” that point to the **FILE**
 11492 objects associated, respectively, with the standard error, input, and output streams:11493 **stderr** Standard error output stream.11494 **stdin** Standard input stream.


```

11544     int      putc_unlocked(int, FILE *);
11545     int      putchar_unlocked(int);
11546     int      puts(const char *);
11547     int      remove(const char *);
11548     int      rename(const char *, const char *);
11549     void     rewind(FILE *);
11550     int      scanf(const char *restrict, ...);
11551     void     setbuf(FILE *restrict, char *restrict);
11552     int      setvbuf(FILE *restrict, char *restrict, int, size_t);
11553     int      snprintf(char *restrict, size_t, const char *restrict, ...);
11554     int      sprintf(char *restrict, const char *restrict, ...);
11555     int      sscanf(const char *restrict, const char *restrict, int ...);
11556 XSI     char  *tempnam(const char *, const char *);
11557     FILE    *tmpfile(void);
11558     char    *tmpnam(char *);
11559     int      ungetc(int, FILE *);
11560     int      vfprintf(FILE *restrict, const char *restrict, va_list);
11561     int      vfscanf(FILE *restrict, const char *restrict, va_list);
11562     int      vprintf(const char *restrict, va_list);
11563     int      vscanf(const char *restrict, va_list);
11564     int      vsnprintf(char *restrict, size_t, const char *restrict, va_list);
11565     int      vsprintf(char *restrict, const char *restrict, va_list);
11566     int      vsscanf(const char *restrict, const char *restrict, va_list arg);

```

11567 XSI **Inclusion of the <stdio.h> header may also make visible all symbols from <stddef.h>.**

11568 APPLICATION USAGE

11569 None.

11570 RATIONALE

11571 None.

11572 FUTURE DIRECTIONS

11573 None.

11574 SEE ALSO

11575 **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *clearerr()*, *ctermid()*,
11576 *fclose()*, *fdopen()*, *fgetc()*, *fgetpos()*, *ferror()*, *feof()*, *fflush()*, *fgets()*, *fileno()*, *flockfile()*, *fopen()*,
11577 *fputc()*, *fputs()*, *fread()*, *freopen()*, *fseek()*, *fsetpos()*, *ftell()*, *fwrite()*, *getc()*, *getc_unlocked()*,
11578 *getwchar()*, *getchar()*, *getopt()*, *gets()*, *pclose()*, *perror()*, *popen()*, *printf()*, *putc()*, *putchar()*, *puts()*,
11579 *putwchar()*, *remove()*, *rename()*, *rewind()*, *scanf()*, *setbuf()*, *setvbuf()*, *sscanf()*, *stdin*, *system()*,
11580 *tempnam()*, *tmpfile()*, *tmpnam()*, *ungetc()*, *vfscanf()*, *vscanf()*, *vprintf()*, *vsscanf()*

11581 CHANGE HISTORY

11582 First released in Issue 1. Derived from Issue 1 of the SVID.

11583 Issue 5

11584 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

11585 Large File System extensions are added.

11586 The constant *L_cuserid* and the external variables *optarg*, *opterr*, *optind*, and *optopt* are marked as
11587 extensions and LEGACY.

11588 The *cuserid()* and *getopt()* functions are marked LEGACY.

11589 **Issue 6**

11590 The constant `L_cuserid` and the external variables `optarg`, `opterr`, `optind`, and `optopt` are removed
11591 as they were previously marked LEGACY.

11592 The `cuserid()`, `getopt()`, and `getw()` functions are removed as they were previously marked |
11593 LEGACY.

11594 Several functions are marked as part of the `_POSIX_THREAD_SAFE_FUNCTIONS` option.

11595 This reference page is updated to align with the ISO/IEC 9899:1999 standard. Note that the
11596 description of the `fpos_t` type is now explicitly updated to exclude array types.

11597 Extensions beyond the ISO C standard are now marked. |

11598 NAME

11599 stdlib.h — standard library definitions

11600 SYNOPSIS

11601 #include <stdlib.h>

11602 DESCRIPTION

11603 cx Some of the functionality described on this reference page extends the ISO C standard.
11604 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
11605 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these
11606 symbols in this header.

11607 The <stdlib.h> header shall define the following macros:

11608 EXIT_FAILURE Unsuccessful termination for *exit()*; evaluates to a non-zero value.

11609 EXIT_SUCCESS Successful termination for *exit()*; evaluates to 0.

11610 NULL Null pointer.

11611 {RAND_MAX} Maximum value returned by *rand()*; at least 32,767.

11612 {MB_CUR_MAX} Integer expression whose value is the maximum number of bytes in a
11613 character specified by the current locale.

11614 The following data types shall be defined through **typedef**:

11615 **div_t** Structure type returned by the *div()* function.

11616 **ldiv_t** Structure type returned by the *ldiv()* function.

11617 **lldiv_t** Structure type returned by the *lldiv()* function.

11618 **size_t** As described in <stddef.h>.

11619 **wchar_t** As described in <stddef.h>.

11620 In addition, the following symbolic names and macros shall be defined as in <sys/wait.h>, for
11621 use in decoding the return value from *system()*:

11622 xsi WNOHANG

11623 WUNTRACED

11624 WEXITSTATUS

11625 WIFEXITED

11626 WIFSIGNALED

11627 WIFSTOPPED

11628 WSTOPSIG

11629 WTERMSIG

11630

11631 The following shall be declared as functions and may also be defined as macros. Function |
11632 prototypes shall be provided. |

11633 void _Exit(int);

11634 xsi long a64l(const char *);

11635 void abort(void);

11636 int abs(int);

11637 int atexit(void (*)(void));

11638 double atof(const char *);

11639 int atoi(const char *);

11640 long atol(const char *);

```

11641     long long    atoll(const char *);
11642     void        *bsearch(const void *, const void *, size_t, size_t,
11643                       int (*)(const void *, const void *));
11644     void        *calloc(size_t, size_t);
11645     div_t       div(int, int);
11646 XSI     double    drand48(void);
11647     char        *ecvt(double, int, int *restrict, int *restrict); (LEGACY)
11648     double     erand48(unsigned short[3]);
11649     void        exit(int);
11650 XSI     char        *fcvt(double, int, int *restrict, int *restrict); (LEGACY)
11651     void        free(void *);
11652 XSI     char        *gcvt(double, int, char *); (LEGACY)
11653     char        *getenv(const char *);
11654 XSI     int        getsubopt(char **, char *const *, char **);
11655     int        grantpt(int);
11656     char        *initstate(unsigned, char *, size_t);
11657     long        jrand48(unsigned short[3]);
11658     char        *l64a(long);
11659     long        labs(long);
11660 XSI     void        lcong48(unsigned short[7]);
11661     ldiv_t      ldiv(long, long);
11662     long long   llabs(long long);
11663     lldiv_t     lldiv(long long, long long);
11664 XSI     long        lrand48(void);
11665     void        *malloc(size_t);
11666     int        mblen(const char *, size_t);
11667     size_t     mbstowcs(wchar_t *restrict, const char *restrict, size_t);
11668     int        mbtowlc(wchar_t *restrict, const char *restrict, size_t);
11669 XSI     char        *mktemp(char *); (LEGACY)
11670     int        mkstemp(char *);
11671     long        mrand48(void);
11672     long        nrand48(unsigned short[3]);
11673 ADV     int        posix_memalign(void **, size_t, size_t);
11674 XSI     int        posix_openpt(int);
11675     char        *ptsname(int);
11676     int        putenv(char *);
11677     void        qsort(void *, size_t, size_t, int (*)(const void *,
11678                       const void *));
11679     int        rand(void);
11680 TSF     int        rand_r(unsigned *);
11681 XSI     long        random(void);
11682     void        *realloc(void *, size_t);
11683 XSI     char        *realpath(const char *restrict, char *restrict);
11684     unsigned short seed48(unsigned short[3]);
11685 CX     int        setenv(const char *, const char *, int);
11686 XSI     void        setkey(const char *);
11687     char        *setstate(const char *);
11688     void        srand(unsigned);
11689 XSI     void        srand48(long);
11690     void        srandom(unsigned);
11691     double     strtod(const char *restrict, char **restrict);
11692     float      strtod(const char *restrict, char **restrict);

```

```

11693     long          strtol(const char *restrict, char **restrict, int);
11694     long double   strtold(const char *restrict, char **restrict);
11695     long long     strtoll(const char *restrict, char **restrict, int);
11696     unsigned long strtoul(const char *restrict, char **restrict, int);
11697     long long     strtoull(const char *restrict, char **restrict, int);
11698     int           system(const char *);
11699 XSI     int         unlockpt(int);
11700 CX     int         unsetenv(const char *);
11701     size_t        wcstombs(char *restrict, const wchar_t *restrict, size_t);
11702     int           wctomb(char *, wchar_t);

11703 XSI     Inclusion of the <stdlib.h> header may also make visible all symbols from <stddef.h>,
11704         <limits.h>, <math.h>, and <sys/wait.h>.

```

11705 APPLICATION USAGE

11706 None.

11707 RATIONALE

11708 None.

11709 FUTURE DIRECTIONS

11710 None.

11711 SEE ALSO

11712 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *_Exit()*, *a64l()*, *abort()*,
11713 *abs()*, *atexit()*, *atof()*, *atoi()*, *atol()*, *atoll()*, *bsearch()*, *calloc()*, *div()*, *drand48()*, *erand48()*, *exit()*,
11714 *free()*, *getenv()*, *getsubopt()*, *grantpt()*, *initstate()*, *jrand48()*, *l64a()*, *labs()*, *lcong48()*, *ldiv()*, *llabs()*,
11715 *lldiv()*, *lrand48()*, *malloc()*, *mblen()*, *mbstowcs()*, *mbtowc()*, *mkstemp()*, *mrand48()*, *nrand48()*,
11716 *posix_memalign()*, *ptsname()*, *putenv()*, *qsort()*, *rand()*, *realloc()*, *realpath()*, *setstate()*, *srand()*,
11717 *srand48()*, *srandom()*, *strtod()*, *strtof()*, *strtol()*, *strtold()*, *strtoll()*, *strtoul()*, *strtoull()*, *unlockpt()*,
11718 *wcstombs()*, *wctomb()*

11719 CHANGE HISTORY

11720 First released in Issue 3.

11721 Issue 5

11722 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

11723 The *ttyslot()* and *valloc()* functions are marked LEGACY.

11724 The type of the third argument to *initstate()* is changed from **int** to **size_t**. The type of the return
11725 value from *setstate()* is changed from **char** to **char ***, and the type of the first argument is
11726 changed from **char *** to **const char ***.

11727 Issue 6

11728 The Open Group Corrigendum U021/1 is applied, correcting the prototype for *realpath()* to be
11729 consistent with the reference page.

11730 The Open Group Corrigendum U028/13 is applied, correcting the prototype for *putenv()* to be
11731 consistent with the reference page.

11732 The *rand_r()* function is marked as part of the `_POSIX_THREAD_SAFE_FUNCTIONS` option.

11733 Function prototypes for *setenv()* and *unsetenv()* are added.

11734 The *posix_memalign()* function is added for alignment with IEEE Std 1003.1d-1999.

11735 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

- 11736 The *ecvt()*, *fcvt()*, *gcvt()*, and *mktemp()* functions are marked LEGACY.
- 11737 The *ttyslot()* and *valloc()* functions are removed as they were previously marked LEGACY. |
- 11738 Extensions beyond the ISO C standard are now marked. |

11739 **NAME**11740 `string.h` — string operations11741 **SYNOPSIS**11742 `#include <string.h>`11743 **DESCRIPTION**

11744 CX Some of the functionality described on this reference page extends the ISO C standard.
 11745 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 11746 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these
 11747 symbols in this header.

11748 The **<string.h>** header shall define the following:11749 **NULL** Null pointer constant.11750 **size_t** As described in **<stddef.h>**.

11751 The following shall be declared as functions and may also be defined as macros. Function |
 11752 prototypes shall be provided. |

11753 XSI `void *memcpy(void *restrict, const void *restrict, int, size_t);`11754 `void *memchr(const void *, int, size_t);`11755 `int memcmp(const void *, const void *, size_t);`11756 `void *memcpy(void *restrict, const void *restrict, size_t);`11757 `void *memmove(void *, const void *, size_t);`11758 `void *memset(void *, int, size_t);`11759 `char *strcat(char *restrict, const char *restrict);`11760 `char *strchr(const char *, int);`11761 `int strcmp(const char *, const char *);`11762 `int strcoll(const char *, const char *);`11763 `char *strcpy(char *restrict, const char *restrict);`11764 `size_t strcspn(const char *, const char *);`11765 XSI `char *strdup(const char *);`11766 `char *strerror(int);`11767 `size_t strlen(const char *);`11768 `char *strncat(char *restrict, const char *restrict, size_t);`11769 `int strncmp(const char *, const char *, size_t);`11770 `char *strncpy(char *restrict, const char *restrict, size_t);`11771 `char *strpbrk(const char *, const char *);`11772 `char *strrchr(const char *, int);`11773 `size_t strspn(const char *, const char *);`11774 `char *strstr(const char *, const char *);`11775 `char *strtok(char *restrict, const char *restrict);`11776 TSF `char *strtok_r(char *, const char *, char **);`11777 `size_t strxfrm(char *restrict, const char *restrict, size_t);`11778 XSI Inclusion of the **<string.h>** header may also make visible all symbols from **<stddef.h>**.

11779 **APPLICATION USAGE**

11780 None.

11781 **RATIONALE**

11782 None.

11783 **FUTURE DIRECTIONS**

11784 None.

11785 **SEE ALSO**

11786 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *memcpy()*, *memchr()*,
11787 *memcmp()*, *memcpy()*, *memmove()*, *memset()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*, *strcpy()*,
11788 *strcspn()*, *strdup()*, *strerror()*, *strlen()*, *strncat()*, *strncmp()*, *strncpy()*, *strpbrk()*, *strrchr()*, *strspn()*,
11789 *strstr()*, *strtok()*, *strxfrm()*

11790 **CHANGE HISTORY**

11791 First released in Issue 1. Derived from Issue 1 of the SVID.

11792 **Issue 5**

11793 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

11794 **Issue 6**11795 The *strtok_r()* function is marked as part of the `_POSIX_THREAD_SAFE_FUNCTIONS` option.

11796 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

11797 **NAME**

11798 strings.h — string operations

11799 **SYNOPSIS**

11800 XSI #include <strings.h>

11801

11802 **DESCRIPTION**11803 The following shall be declared as functions and may also be defined as macros. Function |
11804 prototypes shall be provided. |11805 int bcmp(const void *, const void *, size_t); (**LEGACY**)11806 void bcopy(const void *, void *, size_t); (**LEGACY**)11807 void bzero(void *, size_t); (**LEGACY**)

11808 int ffs(int);

11809 char *index(const char *, int); (**LEGACY**)11810 char *rindex(const char *, int); (**LEGACY**)

11811 int strcasecmp(const char *, const char *);

11812 int strncasecmp(const char *, const char *, size_t);

11813 The `size_t` type shall be defined through `typedef` as described in `<stddef.h>`.11814 **APPLICATION USAGE**

11815 None.

11816 **RATIONALE**

11817 None.

11818 **FUTURE DIRECTIONS**

11819 None.

11820 **SEE ALSO**11821 `<stddef.h>`, the System Interfaces volume of IEEE Std 1003.1-200x, `ffs()`, `strcasecmp()`,
11822 `strncasecmp()`11823 **CHANGE HISTORY**

11824 First released in Issue 4, Version 2.

11825 **Issue 6**11826 The Open Group Corrigendum U021/2 is applied, correcting the prototype for `index()` to be
11827 consistent with the reference page.11828 The `bcmp()`, `bcopy()`, `bzero()`, `index()`, and `rindex()` functions are marked LEGACY.

11829 **NAME**

11830 `stropts.h` — STREAMS interface (**STREAMS**)

11831 **SYNOPSIS**

11832 XSR `#include <stropts.h>`

11833

11834 **DESCRIPTION**

11835 The `<stropts.h>` header shall define the **bandinfo** structure that includes at least the following
 11836 members:

11837	<code>unsigned char</code>	<code>bi_pri</code>	Priority band.	
11838	<code>int</code>	<code>bi_flag</code>	Flushing type.	

11839 The `<stropts.h>` header shall define the **strpeek** structure that includes at least the following
 11840 members:

11841	<code>struct strbuf</code>	<code>ctlbuf</code>	The control portion of the message.	
11842	<code>struct strbuf</code>	<code>databuf</code>	The data portion of the message.	
11843	<code>t_uscalar_t</code>	<code>flags</code>	<code>RS_HIPRI</code> or 0.	

11844 The `<stropts.h>` header shall define the **strbuf** structure that includes at least the following
 11845 members:

11846	<code>int</code>	<code>maxlen</code>	Maximum buffer length.	
11847	<code>int</code>	<code>len</code>	Length of data.	
11848	<code>char</code>	<code>*buf</code>	Pointer to buffer.	

11849 The `<stropts.h>` header shall define the **strfdinsert** structure that includes at least the following
 11850 members:

11851	<code>struct strbuf</code>	<code>ctlbuf</code>	The control portion of the message.	
11852	<code>struct strbuf</code>	<code>databuf</code>	The data portion of the message.	
11853	<code>t_uscalar_t</code>	<code>flags</code>	<code>RS_HIPRI</code> or 0.	
11854	<code>int</code>	<code>fildes</code>	File descriptor of the other STREAM.	
11855	<code>int</code>	<code>offset</code>	Relative location of the stored value.	

11856 The `<stropts.h>` header shall define the **striocctl** structure that includes at least the following
 11857 members:

11858	<code>int</code>	<code>ic_cmd</code>	<code>ioctl()</code> command.	
11859	<code>int</code>	<code>ic_timeout</code>	Timeout for response.	
11860	<code>int</code>	<code>ic_len</code>	Length of data.	
11861	<code>char</code>	<code>*ic_dp</code>	Pointer to buffer.	

11862 The `<stropts.h>` header shall define the **strrecvfd** structure that includes at least the following
 11863 members:

11864	<code>int</code>	<code>fda</code>	Received file descriptor.	
11865	<code>uid_t</code>	<code>uid</code>	UID of sender.	
11866	<code>gid_t</code>	<code>gid</code>	GID of sender.	

11867 The **uid_t** and **gid_t** types shall be defined through **typedef** as described in `<sys/types.h>`.

11868 The `<stropts.h>` header shall define the **t_scalar_t** and **t_uscalar_t** types respectively as signed
 11869 and unsigned opaque types of equal length of at least 32 bits.

11870 The `<stropts.h>` header shall define the **str_list** structure that includes at least the following
 11871 members:

11872	int	sl_nmods	Number of STREAMS module names.		
11873	struct str_mlist	*sl_modlist	STREAMS module names.		
11874	The <stropts.h> header shall define the str_mlist structure that includes at least the following				
11875	member:				
11876	char	l_name[FMNAMESZ+1]	A STREAMS module name.		
11877	At least the following macros shall be defined for use as the <i>request</i> argument to <i>ioctl()</i> :				
11878	I_PUSH		Push a STREAMS module.		
11879	I_POP		Pop a STREAMS module.		
11880	I_LOOK		Get the top module name.		
11881	I_FLUSH		Flush a STREAM.		
11882	I_FLUSHBAND		Flush one band of a STREAM.		
11883	I_SETSIG		Ask for notification signals.		
11884	I_GETSIG		Retrieve current notification signals.		
11885	I_FIND		Look for a STREAMS module.		
11886	I_PEEK		Peek at the top message on a STREAM.		
11887	I_SRDOPT		Set the read mode.		
11888	I_GRDOPT		Get the read mode.		
11889	I_NREAD		Size the top message.		
11890	I_FDINSERT		Send implementation-defined information about another STREAM.		
11891	I_STR		Send a STREAMS <i>ioctl()</i> .		
11892	I_SWROPT		Set the write mode.		
11893	I_GWROPT		Get the write mode.		
11894	I_SENDFD		Pass a file descriptor through a STREAMS pipe.		
11895	I_RECVFD		Get a file descriptor sent via I_SENDFD.		
11896	I_LIST		Get all the module names on a STREAM.		
11897	I_ATMARK		Is the top message “marked”?		
11898	I_CKBAND		See if any messages exist in a band.		
11899	I_GETBAND		Get the band of the top message on a STREAM.		
11900	I_CANPUT		Is a band writable?		
11901	I_SETCLTIME		Set close time delay.		
11902	I_GETCLTIME		Get close time delay.		
11903	I_LINK		Connect two STREAMS.		
11904	I_UNLINK		Disconnect two STREAMS.		
11905	I_PLINK		Persistently connect two STREAMS.		
11906	I_PUNLINK		Dismantle a persistent STREAMS link.		

11907		At least the following macros shall be defined for use with I_LOOK:	
11908	FMNAMESZ	The minimum size in bytes of the buffer referred to by the <i>arg</i> argument.	
11909		At least the following macros shall be defined for use with I_FLUSH:	
11910	FLUSHR	Flush read queues.	
11911	FLUSHW	Flush write queues.	
11912	FLUSHRW	Flush read and write queues.	
11913		At least the following macros shall be defined for use with I_SETSIG:	
11914	S_RDNORM	A normal (priority band set to 0) message has arrived at the head of a	
11915		STREAM head read queue.	
11916	S_RDBAND	A message with a non-zero priority band has arrived at the head of a STREAM	
11917		head read queue.	
11918	S_INPUT	A message, other than a high-priority message, has arrived at the head of a	
11919		STREAM head read queue.	
11920	S_HIPRI	A high-priority message is present on a STREAM head read queue.	
11921	S_OUTPUT	The write queue for normal data (priority band 0) just below the STREAM	
11922		head is no longer full. This notifies the process that there is room on the queue	
11923		for sending (or writing) normal data downstream.	
11924	S_WRNORM	Equivalent to S_OUTPUT.	
11925	S_WRBAND	The write queue for a non-zero priority band just below the STREAM head is	
11926		no longer full.	
11927	S_MSG	A STREAMS signal message that contains the SIGPOLL signal reaches the	
11928		front of the STREAM head read queue.	
11929	S_ERROR	Notification of an error condition reaches the STREAM head.	
11930	S_HANGUP	Notification of a hangup reaches the STREAM head.	
11931	S_BANDURG	When used in conjunction with S_RDBAND, SIGURG is generated instead of	
11932		SIGPOLL when a priority message reaches the front of the STREAM head read	
11933		queue.	
11934		At least the following macros shall be defined for use with I_PEEK:	
11935	RS_HIPRI	Only look for high-priority messages.	
11936		At least the following macros shall be defined for use with I_SRDOPT:	
11937	RNORM	Byte-STREAM mode, the default.	
11938	RMSGD	Message-discard mode.	
11939	RMSGN	Message-nondiscard mode.	
11940	RPROTNORM	Fail <i>read()</i> with [EBADMSG] if a message containing a control part is at the	
11941		front of the STREAM head read queue.	
11942	RPROTDAT	Deliver the control part of a message as data when a process issues a <i>read()</i> .	
11943	RPROTDIS	Discard the control part of a message, delivering any data part, when a	
11944		process issues a <i>read()</i> .	

11945 At least the following macros shall be defined for use with I_SWOPT: |

11946 SNDZERO Send a zero-length message downstream when a *write()* of 0 bytes occurs.

11947 At least the following macros shall be defined for use with I_ATMARK: |

11948 ANYMARK Check if the message is marked.

11949 LASTMARK Check if the message is the last one marked on the queue.

11950 At least the following macros shall be defined for use with I_UNLINK: |

11951 MUXID_ALL Unlink all STREAMs linked to the STREAM associated with *fildev*.

11952 The following macros shall be defined for *getmsg()*, *getpmsg()*, *putmsg()*, and *putpmsg()*: |

11953 MSG_ANY Receive any message.

11954 MSG_BAND Receive message from specified band.

11955 MSG_HIPRI Send/receive high-priority message.

11956 MORECTL More control information is left in message.

11957 MOREDATA More data is left in message.

11958 The **<stropts.h>** header may make visible all of the symbols from **<unistd.h>**.

11959 The following shall be declared as functions and may also be defined as macros. Function |

11960 prototypes shall be provided. |

```

11961 int isastream(int);
11962 int getmsg(int, struct strbuf *restrict, struct strbuf *restrict,
11963 int *restrict);
11964 int getpmsg(int, struct strbuf *restrict, struct strbuf *restrict,
11965 int *restrict, int *restrict);
11966 int ioctl(int, int, ... );
11967 int putmsg(int, const struct strbuf *, const struct strbuf *, int);
11968 int putpmsg(int, const struct strbuf *, const struct strbuf *, int,
11969 int);
11970 int fattach(int, const char *);
11971 int fdetach(const char *);

```

11972 **APPLICATION USAGE**

11973 None.

11974 **RATIONALE**

11975 None.

11976 **FUTURE DIRECTIONS**

11977 None.

11978 **SEE ALSO**

11979 **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *close()*, *fcntl()*, *getmsg()*,

11980 *ioctl()*, *open()*, *pipe()*, *read()*, *poll()*, *putmsg()*, *signal()*, *write()* |

11981 **CHANGE HISTORY**

11982 First released in Issue 4, Version 2.

11983 **Issue 5**

11984 The *flags* member of the **strpeek** and **strfdinsert** structures are changed from type **long** to
11985 **t_uscalar_t**.

11986 **Issue 6**

11987 This header is marked as part of the XSI STREAMS Option Group.

11988 The **restrict** keyword is added to the prototypes for *getmsg()* and *getpmsg()*.

11989 **NAME**

11990 sys/ipc.h — XSI interprocess communication access structure

11991 **SYNOPSIS**

11992 XSI #include <sys/ipc.h>

11993

11994 **DESCRIPTION**

11995 The **<sys/ipc.h>** header is used by three mechanisms for XSI interprocess communication (IPC):
 11996 messages, semaphores, and shared memory. All use a common structure type, **ipc_perm** to pass
 11997 information used in determining permission to perform an IPC operation.

11998 The **ipc_perm** structure shall contain the following members:

11999	uid_t	uid	Owner's user ID.
12000	gid_t	gid	Owner's group ID.
12001	uid_t	cuid	Creator's user ID.
12002	gid_t	cgid	Creator's group ID.
12003	mode_t	mode	Read/write permission.

12004 The **uid_t**, **gid_t**, **mode_t**, and **key_t** types shall be defined as described in **<sys/types.h>**.

12005 Definitions shall be provided for the following constants:

12006 Mode bits:

12007	IPC_CREAT	Create entry if key does not exist.
12008	IPC_EXCL	Fail if key exists.
12009	IPC_NOWAIT	Error if request must wait.

12010 Keys:

12011	IPC_PRIVATE	Private key.
-------	-------------	--------------

12012 Control commands:

12013	IPC_RMID	Remove identifier.
12014	IPC_SET	Set options.
12015	IPC_STAT	Get options.

12016 The following shall be declared as a function and may also be defined as a macro. A function |
 12017 prototype shall be provided. |

12018 key_t ftok(const char *, int);

12019 **APPLICATION USAGE**

12020 None.

12021 **RATIONALE**

12022 None.

12023 **FUTURE DIRECTIONS**

12024 None.

12025 **SEE ALSO**

12026 **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *ftok()*

12027 **CHANGE HISTORY**

12028 First released in Issue 2. Derived from System V Release 2.0.

12029 **NAME**12030 `sys/mman.h` — memory management declarations12031 **SYNOPSIS**12032 `#include <sys/mman.h>`12033 **DESCRIPTION**12034 The **<sys/mman.h>** header shall be supported if the implementation supports at least one of the
12035 following options:

- 12036 MF • The Memory Mapped Files option
- 12037 SHM • The Shared Memory Objects option
- 12038 ML • The Process Memory Locking option
- 12039 MPR • The Memory Protection option
- 12040 TYM • The Typed Memory Objects option
- 12041 SIO • The Synchronized Input and Output option
- 12042 ADV • The Advisory Information option
- 12043 TYM • The Typed Memory Objects option

12044 MC2 If one or more of the Advisory Information, Memory Mapped Files, or Shared Memory Objects
12045 options are supported, the following protection options shall be defined:

- 12046 MC2 `PROT_READ` Page can be read.
- 12047 MC2 `PROT_WRITE` Page can be written.
- 12048 MC2 `PROT_EXEC` Page can be executed.
- 12049 MC2 `PROT_NONE` Page cannot be accessed.

12050 The following *flag* options shall be defined:

- 12051 MF|SHM `MAP_SHARED` Share changes.
- 12052 MF|SHM `MAP_PRIVATE` Changes are private.
- 12053 MF|SHM `MAP_FIXED` Interpret *addr* exactly.

12054 The following flags shall be defined for *msync()*:

- 12055 MF|SIO `MS_ASYNC` Perform asynchronous writes.
- 12056 MF|SIO `MS_SYNC` Perform synchronous writes.
- 12057 MF|SIO `MS_INVALIDATE` Invalidate mappings.

12058 ML The following symbolic constants shall be defined for the *mlockall()* function:

- 12059 ML `MCL_CURRENT` Lock currently mapped pages.
- 12060 ML `MCL_FUTURE` Lock pages that become mapped.

12061 MF|SHM The symbolic constant `MAP_FAILED` shall be defined to indicate a failure from the *mmap()*
12062 function.12063 MC1 If the Advisory Information and either the Memory Mapped Files or Shared Memory Objects
12064 options are supported, values for *advice* used by *posix_madvise()* shall be defined as follows:

- 12065 `POSIX_MADV_NORMAL`
- 12066 The application has no advice to give on its behavior with respect to the specified range. It

12067		is the default characteristic if no advice is given for a range of memory.
12068		POSIX_MADV_SEQUENTIAL
12069		The application expects to access the specified range sequentially from lower addresses to
12070		higher addresses.
12071		POSIX_MADV_RANDOM
12072		The application expects to access the specified range in a random order.
12073		POSIX_MADV_WILLNEED
12074		The application expects to access the specified range in the near future.
12075		POSIX_MADV_DONTNEED
12076		The application expects that it will not access the specified range in the near future.
12077		
12078	TYM	The following flags shall be defined for <i>posix_typed_mem_open()</i> :
12079		POSIX_TYPED_MEM_ALLOCATE
12080		Allocate on <i>mmap()</i> .
12081		POSIX_TYPED_MEM_ALLOCATE_CONTIG
12082		Allocate contiguously on <i>mmap()</i> .
12083		POSIX_TYPED_MEM_MAP_ALLOCATABLE Map on <i>mmap()</i> , without affecting allocatability.
12084		
12085		The mode_t , off_t , and size_t types shall be defined as described in <sys/types.h>.
12086	TYM	The <sys/mman.h> header shall define the structure posix_typed_mem_info , which includes at
12087		least the following member:
12088		<code>size_t posix_tmi_length</code> Maximum length which may be allocated
12089		from a typed memory object.
12090		
12091		The following shall be declared as functions and may also be defined as macros. Function
12092		prototypes shall be provided.
12093	ML	<code>int mlock(const void *, size_t);</code>
12094		<code>int mlockall(int);</code>
12095	MF SHM	<code>void *mmap(void *, size_t, int, int, int, off_t);</code>
12096	MPR	<code>int mprotect(void *, size_t, int);</code>
12097	MF SIO	<code>int msync(void *, size_t, int);</code>
12098	ML	<code>int munlock(const void *, size_t);</code>
12099		<code>int munlockall(void);</code>
12100	MF SHM	<code>int munmap(void *, size_t);</code>
12101	ADV	<code>int posix_madvise(void *, size_t, int);</code>
12102	TYM	<code>int posix_mem_offset(const void *restrict, size_t, off_t *restrict,</code>
12103		<code>size_t *restrict, int *restrict);</code>
12104		<code>int posix_typed_mem_get_info(int, struct posix_typed_mem_info *);</code>
12105		<code>int posix_typed_mem_open(const char *, int, int);</code>
12106	SHM	<code>int shm_open(const char *, int, mode_t);</code>
12107		<code>int shm_unlink(const char *);</code>
12108		

12109 **APPLICATION USAGE**

12110 None.

12111 **RATIONALE**

12112 None.

12113 **FUTURE DIRECTIONS**

12114 None.

12115 **SEE ALSO**

12116 **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *mlock()*, *mlockall()*,
12117 *mmap()*, *mprotect()*, *msync()*, *munlock()*, *munlockall()*, *munmap()*, *posix_mem_offset()*,
12118 *posix_typed_mem_get_info()*, *posix_typed_mem_open()*, *shm_open()*, *shm_unlink()*

12119 **CHANGE HISTORY**

12120 First released in Issue 4, Version 2.

12121 **Issue 5**

12122 Updated for alignment with the POSIX Realtime Extension.

12123 **Issue 6**

12124 The **<sys/mman.h>** header is marked as dependent on support for either the
12125 `_POSIX_MAPPED_FILES`, `_POSIX_MEMLOCK`, or `_POSIX_SHARED_MEMORY` options.

12126 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 12127 • The TYM margin code is added to the list of margin codes for the **<sys/mman.h>** header line,
12128 as well as for other lines.
- 12129 • The `POSIX_TYPED_MEM_ALLOCATE`, `POSIX_TYPED_MEM_ALLOCATE_CONTIG`, and
12130 `POSIX_TYPED_MEM_MAP_ALLOCATABLE` flags are added.
- 12131 • The **posix_tmi_length** structure is added.
- 12132 • The *posix_mem_offset()*, *posix_typed_mem_get_info()*, and *posix_typed_mem_open()* functions
12133 are added.

12134 The **restrict** keyword is added to the prototype for *posix_mem_offset()*.12135 IEEE PASC Interpretation 1003.1 #102 is applied adding the prototype for *posix_madvise()*.

12136 **NAME**

12137 sys/msg.h — XSI message queue structures

12138 **SYNOPSIS**

12139 XSI `#include <sys/msg.h>`

12140

12141 **DESCRIPTION**

12142 The <sys/msg.h> header shall define the following constant and members of the structure
12143 **msqid_ds**.

12144 The following data types shall be defined through **typedef**:

12145 **msgqnum_t** Used for the number of messages in the message queue.

12146 **msglen_t** Used for the number of bytes allowed in a message queue.

12147 These types shall be unsigned integer types that are able to store values at least as large as a type
12148 **unsigned short**.

12149 Message operation flag:

12150 **MSG_NOERROR** No error if big message.

12151 The **msqid_ds** structure shall contain the following members:

12152	<code>struct ipc_perm</code>	<code>msg_perm</code>	Operation permission structure.
12153	<code>msgqnum_t</code>	<code>msg_qnum</code>	Number of messages currently on queue.
12154	<code>msglen_t</code>	<code>msg_qbytes</code>	Maximum number of bytes allowed on queue.
12155	<code>pid_t</code>	<code>msg_lspid</code>	Process ID of last <i>msgsnd()</i> .
12156	<code>pid_t</code>	<code>msg_lrpid</code>	Process ID of last <i>msgrcv()</i> .
12157	<code>time_t</code>	<code>msg_stime</code>	Time of last <i>msgsnd()</i> .
12158	<code>time_t</code>	<code>msg_rtime</code>	Time of last <i>msgrcv()</i> .
12159	<code>time_t</code>	<code>msg_ctime</code>	Time of last change.

12160 The **pid_t**, **time_t**, **key_t**, **size_t**, and **ssize_t** types shall be defined as described in <sys/types.h>.

12161 The following shall be declared as functions and may also be defined as macros. Function
12162 prototypes shall be provided. |

```
12163 int      msgctl(int, int, struct msqid_ds *);
12164 int      msgget(key_t, int);
12165 ssize_t  msgrcv(int, void *, size_t, long, int);
12166 int      msgsnd(int, const void *, size_t, int);
```

12167 In addition, all of the symbols from <sys/ipc.h> shall be defined when this header is included.

12168 **APPLICATION USAGE**

12169 None.

12170 **RATIONALE**

12171 None.

12172 **FUTURE DIRECTIONS**

12173 None.

12174 **SEE ALSO**

12175 <sys/types.h>, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

12176 **CHANGE HISTORY**

12177 First released in Issue 2. Derived from System V Release 2.0.

12178 **NAME**

12179 sys/resource.h — definitions for XSI resource operations

12180 **SYNOPSIS**

12181 XSI #include <sys/resource.h>

12182

12183 **DESCRIPTION**

12184 The <sys/resource.h> header shall define the following symbolic constants as possible values of
 12185 the *which* argument of *getpriority()* and *setpriority()*:

12186 PRIO_PROCESS Identifies the *who* argument as a process ID.

12187 PRIO_PGRP Identifies the *who* argument as a process group ID.

12188 PRIO_USER Identifies the *who* argument as a user ID.

12189 The following type shall be defined through **typedef**:

12190 **rlim_t** Unsigned integer type used for limit values.

12191 The following symbolic constants shall be defined:

12192 RLIM_INFINITY A value of **rlim_t** indicating no limit.

12193 RLIM_SAVED_MAX A value of type **rlim_t** indicating an unrepresentable saved hard
 12194 limit.

12195 RLIM_SAVED_CUR A value of type **rlim_t** indicating an unrepresentable saved soft limit.

12196 On implementations where all resource limits are representable in an object of type **rlim_t**,
 12197 RLIM_SAVED_MAX and RLIM_SAVED_CUR need not be distinct from RLIM_INFINITY.

12198 The following symbolic constants shall be defined as possible values of the *who* parameter of
 12199 *getrusage()*:

12200 RUSAGE_SELF Returns information about the current process.

12201 RUSAGE_CHILDREN Returns information about children of the current process.

12202 The <sys/resource.h> header shall define the **rlimit** structure that includes at least the following
 12203 members:

12204 rlim_t rlim_cur The current (soft) limit. |

12205 rlim_t rlim_max The hard limit. |

12206 The <sys/resource.h> header shall define the **rusage** structure that includes at least the following
 12207 members: |

12208 struct timeval ru_utime User time used. |

12209 struct timeval ru_stime System time used. |

12210 The **timeval** structure shall be defined as described in <sys/time.h>. |

12211 The following symbolic constants shall be defined as possible values for the *resource* argument of
 12212 *getrlimit()* and *setrlimit()*:

12213 RLIMIT_CORE Limit on size of core dump file.

12214 RLIMIT_CPU Limit on CPU time per process.

12215 RLIMIT_DATA Limit on data segment size.

12216 RLIMIT_FSIZE Limit on file size.

- 12217 RLIMIT_NOFILE Limit on number of open files.
- 12218 RLIMIT_STACK Limit on stack size.
- 12219 RLIMIT_AS Limit on address space size.
- 12220 The following shall be declared as functions and may also be defined as macros. Function |
- 12221 prototypes shall be provided. |
- 12222 int getpriority(int, id_t);
- 12223 int getrlimit(int, struct rlimit *);
- 12224 int getrusage(int, struct rusage *);
- 12225 int setpriority(int, id_t, int);
- 12226 int setrlimit(int, const struct rlimit *);
- 12227 The **id_t** type shall be defined through **typedef** as described in <sys/types.h>.
- 12228 Inclusion of the <sys/resource.h> header may also make visible all symbols from <sys/time.h>.
- 12229 **APPLICATION USAGE**
- 12230 None.
- 12231 **RATIONALE**
- 12232 None.
- 12233 **FUTURE DIRECTIONS**
- 12234 None.
- 12235 **SEE ALSO**
- 12236 <sys/time.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *getpriority()*,
- 12237 *getrusage()*, *getrlimit()*
- 12238 **CHANGE HISTORY**
- 12239 First released in Issue 4, Version 2.
- 12240 **Issue 5**
- 12241 Large File System extensions are added.

12242 **NAME**

12243 sys/select.h — select types

12244 **SYNOPSIS**

12245 #include <sys/select.h>

12246 **DESCRIPTION**

12247 The <sys/select.h> header shall define the **timeval** structure that includes at least the following
 12248 members:

12249 time_t tv_sec Seconds.

12250 suseconds_t tv_usec Microseconds.

12251 The **time_t** and **suseconds_t** types shall be defined as described in <sys/types.h>.

12252 The **sigset_t** type shall be defined as described in <signal.h>.

12253 The **timespec** structure shall be defined as described in <time.h>.

12254 The <sys/select.h> header shall define the **fd_set** type as a structure. |

12255 Each of the following may be declared as a function, or defined as a macro, or both:

12256 void *FD_CLR*(int *fd*, fd_set **fdset*)

12257 Clears the bit for the file descriptor *fd* in the file descriptor set *fdset*.

12258 int *FD_ISSET*(int *fd*, fd_set **fdset*)

12259 Returns a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set by
 12260 *fdset*, and 0 otherwise.

12261 void *FD_SET*(int *fd*, fd_set **fdset*)

12262 Sets the bit for the file descriptor *fd* in the file descriptor set *fdset*.

12263 void *FD_ZERO*(fd_set **fdset*)

12264 Initializes the file descriptor set *fdset* to have zero bits for all file descriptors.

12265 If implemented as macros, these may evaluate their arguments more than once, so applications
 12266 should ensure that the arguments they supply are never expressions with side effects.

12267 The following shall be defined as a macro:

12268 **FD_SETSIZE**

12269 Maximum number of file descriptors in an **fd_set** structure.

12270 The following shall be declared as functions and may also be defined as macros. Function |
 12271 prototypes shall be provided. |

12272 int pselect(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
 12273 const struct timespec *restrict, const sigset_t *restrict);

12274 int select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
 12275 struct timeval *restrict);

12276 Inclusion of the <sys/select.h> header may make visible all symbols from the headers
 12277 <signal.h>, <sys/time.h>, and <time.h>.

12278 **APPLICATION USAGE**

12279 None.

12280 **RATIONALE**

12281 None.

12282 **FUTURE DIRECTIONS**

12283 None.

12284 **SEE ALSO**

12285 <signal.h>, <sys/time.h>, <sys/types.h>, <time.h>, the System Interfaces volume of

12286 IEEE Std 1003.1-200x, *pselect()*, *select()*

12287 **CHANGE HISTORY**

12288 First released in Issue 6. Derived from IEEE Std 1003.1g-2000. |

12289 The requirement for the **fd_set** structure to have a member *fds_bits* has been removed as per The |

12290 Open Group Base Resolution bwg2001-005. |

12291 **NAME**

12292 sys/sem.h — XSI semaphore facility

12293 **SYNOPSIS**

12294 XSI #include <sys/sem.h>

12295

12296 **DESCRIPTION**

12297 The <sys/sem.h> header shall define the following constants and structures.

12298 Semaphore operation flags:

12299 SEM_UNDO Set up adjust on exit entry.

12300 Command definitions for the *semctl()* function shall be provided as follows:

12301 GETNCNT Get *semncnt*.

12302 GETPID Get *sempid*.

12303 GETVAL Get *semval*.

12304 GETALL Get all cases of *semval*.

12305 GETZCNT Get *semzcnt*.

12306 SETVAL Set *semval*.

12307 SETALL Set all cases of *semval*.

12308 The **semid_ds** structure shall contain the following members:

12309 struct ipc_perm sem_perm Operation permission structure.

12310 unsigned short sem_nsems Number of semaphores in set.

12311 time_t sem_otime Last *semop()* time.

12312 time_t sem_ctime Last time changed by *semctl()*.

12313 The **pid_t**, **time_t**, **key_t**, and **size_t** types shall be defined as described in <sys/types.h>.

12314 A semaphore shall be represented by an anonymous structure containing the following
12315 members:

12316 unsigned short semval Semaphore value.

12317 pid_t sempid Process ID of last operation.

12318 unsigned short semncnt Number of processes waiting for *semval*
12319 to become greater than current value.

12320 unsigned short semzcnt Number of processes waiting for *semval*
12321 to become 0.

12322 The **sembuf** structure shall contain the following members:

12323 unsigned short sem_num Semaphore number.

12324 short sem_op Semaphore operation.

12325 short sem_flg Operation flags.

12326 The following shall be declared as functions and may also be defined as macros. Function |
12327 prototypes shall be provided. |

12328 int semctl(int, int, int, ...);

12329 int semget(key_t, int, int);

12330 int semop(int, struct sembuf *, size_t);

12331 In addition, all of the symbols from <sys/ipc.h> shall be defined when this header is included.

12332 **APPLICATION USAGE**

12333 None.

12334 **RATIONALE**

12335 None.

12336 **FUTURE DIRECTIONS**

12337 None.

12338 **SEE ALSO**

12339 <sys/types.h>, *semctl()*, *semget()*, *semop()*

12340 **CHANGE HISTORY**

12341 First released in Issue 2. Derived from System V Release 2.0.

12342 **NAME**

12343 sys/shm.h — XSI shared memory facility

12344 **SYNOPSIS**

12345 XSI `#include <sys/shm.h>`

12346

12347 **DESCRIPTION**

12348 The <sys/shm.h> header shall define the following symbolic constants:

12349 SHM_RDONLY Attach read-only (else read-write).

12350 SHM_RND Round attach address to SHMLBA.

12351 The <sys/shm.h> header shall define the following symbolic value:

12352 SHMLBA Segment low boundary address multiple.

12353 The following data types shall be defined through **typedef**:

12354 **shmatt_t** Unsigned integer used for the number of current attaches that must be able to
12355 store values at least as large as a type **unsigned short**.

12356 The **shmid_ds** structure shall contain the following members:

12357	struct ipc_perm	shm_perm	Operation permission structure.
12358	size_t	shm_segsz	Size of segment in bytes.
12359	pid_t	shm_lpid	Process ID of last shared memory operation.
12360	pid_t	shm_cpid	Process ID of creator.
12361	shmatt_t	shm_nattch	Number of current attaches.
12362	time_t	shm_atime	Time of last <i>shmat()</i> .
12363	time_t	shm_dtime	Time of last <i>shmdt()</i> .
12364	time_t	shm_ctime	Time of last change by <i>shmctl()</i> .

12365 The **pid_t**, **time_t**, **key_t**, and **size_t** types shall be defined as described in <sys/types.h>.

12366 The following shall be declared as functions and may also be defined as macros. Function |
12367 prototypes shall be provided. |

```
12368 void *shmat(int, const void *, int);
12369 int shmctl(int, int, struct shmid_ds *);
12370 int shmdt(const void *);
12371 int shmget(key_t, size_t, int);
```

12372 In addition, all of the symbols from <sys/ipc.h> shall be defined when this header is included.

12373 **APPLICATION USAGE**

12374 None.

12375 **RATIONALE**

12376 None.

12377 **FUTURE DIRECTIONS**

12378 None.

12379 **SEE ALSO**

12380 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *shmat()*, *shmctl()*, *shmdt()*,
12381 *shmget()*

12382 **CHANGE HISTORY**

12383 First released in Issue 2. Derived from System V Release 2.0.

12384 **Issue 5**

12385 The type of *shm_segsz* is changed from **int** to **size_t**.

12386 **NAME**

12387 sys/socket.h — main sockets header

12388 **SYNOPSIS**

12389 #include <sys/socket.h>

12390 **DESCRIPTION**

12391 The <sys/socket.h> header shall define the type **socklen_t**, which is an integer type of width of
 12392 at least 32 bits; see APPLICATION USAGE.

12393 The <sys/socket.h> header shall define the unsigned integer type **sa_family_t**.

12394 The <sys/socket.h> header shall define the **sockaddr** structure that includes at least the
 12395 following members:

12396 sa_family_t sa_family Address family.
 12397 char sa_data[] Socket address (variable-length data).

12398 The **sockaddr** structure is used to define a socket address which is used in the *bind()*, *connect()*,
 12399 *getpeername()*, *getsockname()*, *recvfrom()*, and *sendto()* functions.

12400 The <sys/socket.h> header shall define the **sockaddr_storage** structure. This structure shall be:

- 12401 • Large enough to accommodate all supported protocol-specific address structures
- 12402 • Aligned at an appropriate boundary so that pointers to it can be cast as pointers to protocol-
 12403 specific address structures and used to access the fields of those structures without
 12404 alignment problems

12405 The **sockaddr_storage** structure shall contain at least the following members:

12406 sa_family_t ss_family

12407 When a **sockaddr_storage** structure is cast as a **sockaddr** structure, the *ss_family* field of the
 12408 **sockaddr_storage** structure shall map onto the *sa_family* field of the **sockaddr** structure. When a
 12409 **sockaddr_storage** structure is cast as a protocol-specific address structure, the *ss_family* field
 12410 shall map onto a field of that structure that is of type **sa_family_t** and that identifies the
 12411 protocol's address family.

12412 The <sys/socket.h> header shall define the **msghdr** structure that includes at least the following
 12413 members:

12414 void *msg_name Optional address.
 12415 socklen_t msg_namelen Size of address.
 12416 struct iovec *msg_iov Scatter/gather array.
 12417 int msg_iovlen Members in msg_iov.
 12418 void *msg_control Ancillary data; see below.
 12419 socklen_t msg_controllen Ancillary data buffer len.
 12420 int msg_flags Flags on received message.

12421 The **msghdr** structure is used to minimize the number of directly supplied parameters to the
 12422 *recvmsg()* and *sendmsg()* functions. This structure is used as a *value-result* parameter in the
 12423 *recvmsg()* function and *value* only for the *sendmsg()* function.

12424 The **iovec** structure shall be defined as described in <sys/uo.h>.

12425 The <sys/socket.h> header shall define the **cmsghdr** structure that includes at least the following
 12426 members:

12427 socklen_t cmsg_len Data byte count, including the cmsghdr.
 12428 int cmsg_level Originating protocol.

12429 int msg_type Protocol-specific type.

12430 The **msg_hdr** structure is used for storage of ancillary data object information.

12431 Ancillary data consists of a sequence of pairs, each consisting of a **msg_hdr** structure followed
12432 by a data array. The data array contains the ancillary data message, and the **msg_hdr** structure
12433 contains descriptive information that allows an application to correctly parse the data.

12434 The values for *msg_level* shall be legal values for the *level* argument to the *getsockopt()* and
12435 *setsockopt()* functions. The system documentation shall specify the *msg_type* definitions for the
12436 supported protocols.

12437 Ancillary data is also possible at the socket level. The <sys/socket.h> header defines the
12438 following macro for use as the *msg_type* value when *msg_level* is SOL_SOCKET:

12439 SCM_RIGHTS Indicates that the data array contains the access rights to be sent or
12440 received.

12441 The <sys/socket.h> header defines the following macros to gain access to the data arrays in the
12442 ancillary data associated with a message header:

12443 MSG_DATA(*msg*)
12444 If the argument is a pointer to a **msg_hdr** structure, this macro shall return an unsigned
12445 character pointer to the data array associated with the **msg_hdr** structure.

12446 MSG_NXTHDR(*mhdr, msg*)
12447 If the first argument is a pointer to a **msg_hdr** structure and the second argument is a pointer
12448 to a **msg_hdr** structure in the ancillary data pointed to by the *msg_control* field of that
12449 **msg_hdr** structure, this macro shall return a pointer to the next **msg_hdr** structure, or a null
12450 pointer if this structure is the last **msg_hdr** in the ancillary data.

12451 MSG_FIRSTHDR(*mhdr*)
12452 If the argument is a pointer to a **msg_hdr** structure, this macro shall return a pointer to the
12453 first **msg_hdr** structure in the ancillary data associated with this **msg_hdr** structure, or a null
12454 pointer if there is no ancillary data associated with the **msg_hdr** structure.

12455 The <sys/socket.h> header shall define the **linger** structure that includes at least the following
12456 members:

12457 int l_onoff Indicates whether linger option is enabled.
12458 int l_linger Linger time, in seconds.

12459 The <sys/socket.h> header shall define the following macros, with distinct integer values:

12460 SOCK_DGRAM Datagram socket. |

12461 RS SOCK_RAW Raw Protocol Interface. |

12462 SOCK_SEQPACKET Sequenced-packet socket. |

12463 SOCK_STREAM Byte-stream socket. |

12464 The <sys/socket.h> header shall define the following macro for use as the *level* argument of
12465 *setsockopt()* and *getsockopt()*.

12466 SOL_SOCKET Options to be accessed at socket level, not protocol level.

12467 The <sys/socket.h> header shall define the following macros, with distinct integer values, for
12468 use as the *option_name* argument in *getsockopt()* or *setsockopt()* calls:

12469 SO_ACCEPTCONN Socket is accepting connections.

12470	SO_BROADCAST	Transmission of broadcast messages is supported.	
12471	SO_DEBUG	Debugging information is being recorded.	
12472	SO_DONTROUTE	Bypass normal routing.	
12473	SO_ERROR	Socket error status.	
12474	SO_KEEPALIVE	Connections are kept alive with periodic messages.	
12475	SO_LINGER	Socket lingers on close.	
12476	SO_OOBINLINE	Out-of-band data is transmitted in line.	
12477	SO_RCVBUF	Receive buffer size.	
12478	SO_RCVLOWAT	Receive “low water mark”.	
12479	SO_RCVTIMEO	Receive timeout.	
12480	SO_REUSEADDR	Reuse of local addresses is supported.	
12481	SO_SNDBUF	Send buffer size.	
12482	SO_SNDLOWAT	Send “low water mark”.	
12483	SO_SNDTIMEO	Send timeout.	
12484	SO_TYPE	Socket type.	
12485	The <sys/socket.h> header shall define the following macro as the maximum <i>backlog</i> queue		
12486	length which may be specified by the <i>backlog</i> field of the <i>listen()</i> function:		
12487	SOMAXCONN	The maximum <i>backlog</i> queue length.	
12488	The <sys/socket.h> header shall define the following macros, with distinct integer values, for		
12489	use as the valid values for the <i>msg_flags</i> field in the msghdr structure, or the <i>flags</i> parameter in		
12490	<i>recvfrom()</i> , <i>recvmsg()</i> , <i>sendmsg()</i> , or <i>sendto()</i> calls:		
12491	MSG_TRUNC	Control data truncated.	
12492	MSG_DONTROUTE	Send without using routing tables.	
12493	MSG_EOR	Terminates a record (if supported by the protocol).	
12494	MSG_OOB	Out-of-band data.	
12495	MSG_PEEK	Leave received data in queue.	
12496	MSG_TRUNC	Normal data truncated.	
12497	MSG_WAITALL	Attempt to fill the read buffer.	
12498	The <sys/socket.h> header shall define the following macros, with distinct integer values:		
12499	AF_INET	Internet domain sockets for use with IPv4 addresses.	
12500	IP6 AF_INET6	Internet domain sockets for use with IPv6 addresses.	
12501	AF_UNIX	UNIX domain sockets.	
12502	AF_UNSPEC	Unspecified .	
12503	The <sys/socket.h> header shall define the following macros, with distinct integer values:		
12504	SHUT_RD	Disables further receive operations.	

12505 SHUT_RDWR Disables further send and receive operations. |
 12506 SHUT_WR Disables further send operations. |
 12507 The following shall be declared as functions and may also be defined as macros. Function |
 12508 prototypes shall be provided. |

```

12509 int accept(int, struct sockaddr *restrict, socklen_t *restrict);
12510 int bind(int, const struct sockaddr *, socklen_t);
12511 int connect(int, const struct sockaddr *, socklen_t);
12512 int getpeername(int, struct sockaddr *restrict, socklen_t *restrict);
12513 int getsockname(int, struct sockaddr *restrict, socklen_t *restrict);
12514 int getsockopt(int, int, int, void *restrict, socklen_t *restrict);
12515 int listen(int, int);
12516 ssize_t recv(int, void *, size_t, int);
12517 ssize_t recvfrom(int, void *restrict, size_t, int,
12518 struct sockaddr *restrict, socklen_t *restrict);
12519 ssize_t recvmsg(int, struct msghdr *, int);
12520 ssize_t send(int, const void *, size_t, int);
12521 ssize_t sendmsg(int, const struct msghdr *, int);
12522 ssize_t sendto(int, const void *, size_t, int, const struct sockaddr *,
12523 socklen_t);
12524 int setsockopt(int, int, int, const void *, socklen_t);
12525 int shutdown(int, int);
12526 int socket(int, int, int);
12527 int socketpair(int, int, int, int[2]);
  
```

12528 Inclusion of <sys/socket.h> may also make visible all symbols from <sys/unistd.h>. |

12529 **APPLICATION USAGE**

12530 To forestall portability problems, it is recommended that applications not use values larger than |
 12531 $2^{31} - 1$ for the **socklen_t** type. |

12532 The **sockaddr_storage** structure solves the problem of declaring storage for automatic variables |
 12533 which is both large enough and aligned enough for storing the socket address data structure of |
 12534 any family. For example, code with a file descriptor and without the context of the address |
 12535 family can pass a pointer to a variable of this type, where a pointer to a socket address structure |
 12536 is expected in calls such as *getpeername()*, and determine the address family by accessing the |
 12537 received content after the call.

12538 The example below illustrates a data structure which aligns on a 64-bit boundary. An |
 12539 implementation-defined field *_ss_align* following *_ss_pad1* is used to force a 64-bit alignment |
 12540 which covers proper alignment good enough for needs of at least **sockaddr_in6** (IPv6) and |
 12541 **sockaddr_in** (IPv4) address data structures. The size of padding field *_ss_pad1* depends on the |
 12542 chosen alignment boundary. The size of padding field *_ss_pad2* depends on the value of overall |
 12543 size chosen for the total size of the structure. This size and alignment are represented in the |
 12544 above example by implementation-defined (not required) constants *_SS_MAXSIZE* (chosen |
 12545 value 128) and *_SS_ALIGNMENT* (with chosen value 8). Constants *_SS_PAD1SIZE* (derived |
 12546 value 6) and *_SS_PAD2SIZE* (derived value 112) are also for illustration and not required. The |
 12547 implementation-defined definitions and structure field names above start with an underscore to |
 12548 denote implementation private name space. Portable code is not expected to access or reference |
 12549 those fields or constants. |

```

12550 /*
12551  * Desired design of maximum size and alignment.
12552  */
  
```

```

12553     #define _SS_MAXSIZE 128
12554         /* Implementation-defined maximum size. */
12555     #define _SS_ALIGNSIZE (sizeof(int64_t))
12556         /* Implementation-defined desired alignment. */
12557
12557     /*
12558      * Definitions used for sockaddr_storage structure paddings design.
12559      */
12560     #define _SS_PAD1SIZE (_SS_ALIGNSIZE - sizeof(sa_family_t))
12561     #define _SS_PAD2SIZE (_SS_MAXSIZE - (sizeof(sa_family_t)+
12562         _SS_PAD1SIZE + _SS_ALIGNSIZE))
12563     struct sockaddr_storage {
12564         sa_family_t  ss_family; /* Address family. */
12565     /*
12566      * Following fields are implementation-defined. */
12567     /*
12568         char _ss_pad1[_SS_PAD1SIZE];
12569         /* 6-byte pad; this is to make implementation-defined
12570          pad up to alignment field that follows explicit in
12571          the data structure. */
12572         int64_t  _ss_align; /* Field to force desired structure
12573          storage alignment. */
12574         char _ss_pad2[_SS_PAD2SIZE];
12575         /* 112-byte pad to achieve desired size,
12576          _SS_MAXSIZE value minus size of ss_family
12577          __ss_pad1, __ss_align fields is 112. */
12578     };
12579 RATIONALE
12580     None.
12581 FUTURE DIRECTIONS
12582     None.
12583 SEE ALSO
12584     <sys/uid.h>, the System Interfaces volume of IEEE Std 1003.1-200x, accept(), bind(), connect(),
12585     getpeername(), getsockname(), getsockopt(), listen(), recv(), recvfrom(), recvmsg(), send(),
12586     sendmsg(), sendto(), setsockopt(), shutdown(), socket(), socketpair()
12587 CHANGE HISTORY
12588     First released in Issue 6. Derived from the XNS, Issue 5.2 specification.
12589     The restrict keyword is added to the prototypes for accept(), getpeername(), getsockname(),
12590     getsockopt(), and recvfrom().

```

12591 NAME

12592 sys/stat.h — data returned by the stat() function

12593 SYNOPSIS

12594 #include <sys/stat.h>

12595 DESCRIPTION

12596 The <sys/stat.h> header shall define the structure of the data returned by the functions *fstat()*,
12597 *lstat()*, and *stat()*.

12598 The **stat** structure shall contain at least the following members:

12599	dev_t	st_dev	ID of device containing file.
12600	ino_t	st_ino	File serial number.
12601	mode_t	st_mode	Mode of file (see below).
12602	nlink_t	st_nlink	Number of hard links to the file.
12603	uid_t	st_uid	User ID of file.
12604	gid_t	st_gid	Group ID of file.
12605 XSI	dev_t	st_rdev	Device ID (if file is character or block special).
12606	off_t	st_size	For regular files, the file size in bytes.
12607			For symbolic links, the length in bytes of the
12608			pathname contained in the symbolic link.
12609 SHM			For a shared memory object, the length in bytes.
12610 TYM			For a typed memory object, the length in bytes.
12611			For other file types, the use of this field is
12612			unspecified
12613	time_t	st_atime	Time of last access.
12614	time_t	st_mtime	Time of last data modification.
12615	time_t	st_ctime	Time of last status change.
12616 XSI	blksize_t	st_blksize	A file system-specific preferred I/O block size for
12617			this object. In some file system types, this may
12618			vary from file to file.
12619	blkcnt_t	st_blocks	Number of blocks allocated for this object.
12620			

12621 File serial number and device ID taken together uniquely identify the file within the system. The
12622 **blkcnt_t**, **blksize_t**, **dev_t**, **ino_t**, **mode_t**, **nlink_t**, **uid_t**, **gid_t**, **off_t**, and **time_t** types shall be
12623 defined as described in <sys/types.h>. Times shall be given in seconds since the Epoch.

12624 Unless otherwise specified, the structure members *st_mode*, *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atime*,
12625 *st_ctime*, and *st_mtime* shall have meaningful values for all file types defined in
12626 IEEE Std 1003.1-200x.

12627 For symbolic links, the *st_mode* member shall contain meaningful information, which can be
12628 used with the file type macros described below, that take a *mode* argument. The *st_size* member
12629 shall contain the length, in bytes, of the pathname contained in the symbolic link. File mode bits
12630 and the contents of the remaining members of the **stat** structure are unspecified. The value
12631 returned in the *st_size* field shall be the length of the contents of the symbolic link, and shall not
12632 count a trailing null if one is present.

12633 The following symbolic names for the values of type *mode_t* shall also be defined.

12634 File type:

12635 XSI	S_IFMT	Type of file.
12636	S_IFBLK	Block special.

12637		S_IFCHR	Character special.
12638		S_IFIFO	FIFO special.
12639		S_IFREG	Regular.
12640		S_IFDIR	Directory.
12641		S_IFLNK	Symbolic link.
12642		S_IFSOCK	Socket.
12643		File mode bits:	
12644	S_IRWXU	Read, write, execute/search by owner.	
12645		S_IRUSR	Read permission, owner.
12646		S_IWUSR	Write permission, owner.
12647		S_IXUSR	Execute/search permission, owner.
12648	S_IRWXG	Read, write, execute/search by group.	
12649		S_IRGRP	Read permission, group.
12650		S_IWGRP	Write permission, group.
12651		S_IXGRP	Execute/search permission, group.
12652	S_IRWXO	Read, write, execute/search by others.	
12653		S_IROTH	Read permission, others.
12654		S_IWOTH	Write permission, others.
12655		S_IXOTH	Execute/search permission, others.
12656	S_ISUID	Set-user-ID on execution.	
12657	S_ISGID	Set-group-ID on execution.	
12658	XSI	S_ISVTX	On directories, restricted deletion flag.
12659		The bits defined by S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH,	
12660	XSI	S_IWOTH, S_IXOTH, S_ISUID, S_ISGID, and S_ISVTX shall be unique.	
12661		S_IRWXU is the bitwise-inclusive OR of S_IRUSR, S_IWUSR, and S_IXUSR.	
12662		S_IRWXG is the bitwise-inclusive OR of S_IRGRP, S_IWGRP, and S_IXGRP.	
12663		S_IRWXO is the bitwise-inclusive OR of S_IROTH, S_IWOTH, and S_IXOTH.	
12664		Implementations may OR other implementation-defined bits into S_IRWXU, S_IRWXG, and	
12665		S_IRWXO, but they shall not overlap any of the other bits defined in this volume of	
12666		IEEE Std 1003.1-200x. The <i>file permission bits</i> are defined to be those corresponding to the	
12667		bitwise-inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO.	
12668		The following macros shall be provided to test whether a file is of the specified type. The value	
12669		<i>m</i> supplied to the macros is the value of <i>st_mode</i> from a stat structure. The macro shall evaluate	
12670		to a non-zero value if the test is true; 0 if the test is false.	
12671	S_ISBLK(<i>m</i>)	Test for a block special file.	
12672	S_ISCHR(<i>m</i>)	Test for a character special file.	

12673 **S_ISDIR(*m*)** Test for a directory.

12674 **S_ISFIFO(*m*)** Test for a pipe or FIFO special file.

12675 **S_ISREG(*m*)** Test for a regular file.

12676 **S_ISLNK(*m*)** Test for a symbolic link.

12677 **S_ISSOCK(*m*)** Test for a socket.

12678 The implementation may implement message queues, semaphores, or shared memory objects as distinct file types. The following macros shall be provided to test whether a file is of the specified type. The value of the *buf* argument supplied to the macros is a pointer to a **stat** structure. The macro shall evaluate to a non-zero value if the specified object is implemented as a distinct file type and the specified file type is contained in the **stat** structure referenced by *buf*. Otherwise, the macro shall evaluate to zero.

12684 **S_TYPEISMQ(*buf*)** Test for a message queue.

12685 **S_TYPEISSEM(*buf*)** Test for a semaphore.

12686 **S_TYPEISSHM(*buf*)** Test for a shared memory object.

12687 TYP 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. The value of the *buf* argument supplied to the macros is a pointer to a **stat** structure. The macro shall evaluate to a non-zero value if the specified object is implemented as a distinct file type and the specified file type is contained in the **stat** structure referenced by *buf*. Otherwise, the macro shall evaluate to zero.

12692 **S_TYPEISTMO(*buf*)** Test macro for a typed memory object.

12694 The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

12696 `int chmod(const char *, mode_t);`

12697 `int fchmod(int, mode_t);`

12698 `int fstat(int, struct stat *);`

12699 `int lstat(const char *restrict, struct stat *restrict);`

12700 `int mkdir(const char *, mode_t);`

12701 `int mkfifo(const char *, mode_t);`

12702 XSI `int mknod(const char *, mode_t, dev_t);`

12703 `int stat(const char *restrict, struct stat *restrict);`

12704 `mode_t umask(mode_t);`

12705 APPLICATION USAGE

12706 Use of the macros is recommended for determining the type of a file.

12707 RATIONALE

12708 A conforming C-language application must include **<sys/stat.h>** for functions that have arguments or return values of type **mode_t**, so that symbolic values for that type can be used. An alternative would be to require that these constants are also defined by including **<sys/types.h>**.

12712 The **S_ISUID** and **S_ISGID** bits may be cleared on any write, not just on *open()*, as some historical implementations do it.

12714 System calls that update the time entry fields in the **stat** structure must be documented by the implementors. POSIX-conforming systems should not update the time entry fields for functions listed in the System Interfaces volume of IEEE Std 1003.1-200x unless the standard requires that

- 12717 they do, except in the case of documented extensions to the standard.
- 12718 Note that *st_dev* must be unique within a Local Area Network (LAN) in a “system” made up of
12719 multiple computers’ file systems connected by a LAN.
- 12720 Networked implementations of a POSIX-conforming system must guarantee that all files visible
12721 within the file tree (including parts of the tree that may be remotely mounted from other
12722 machines on the network) on each individual processor are uniquely identified by the
12723 combination of the *st_ino* and *st_dev* fields.
- 12724 **FUTURE DIRECTIONS**
- 12725 No new *S_IFMT* symbolic names for the file type values of **mode_t** will be defined by
12726 IEEE Std 1003.1-200x; if new file types are required, they will only be testable through *S_ISxx()*
12727 macros instead.
- 12728 **SEE ALSO**
- 12729 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *chmod()*, *fchmod()*, *fstat()*,
12730 *lstat()*, *mkdir()*, *mkfifo()*, *mknod()*, *stat()*, *umask()*
- 12731 **CHANGE HISTORY**
- 12732 First released in Issue 1. Derived from Issue 1 of the SVID.
- 12733 **Issue 5**
- 12734 The DESCRIPTION is updated for alignment with POSIX Realtime Extension.
- 12735 The type of *st_blksize* is changed from **long** to **blksize_t**; the type of *st_blocks* is changed from
12736 **long** to **blkcnt_t**.
- 12737 **Issue 6**
- 12738 The *S_TYPEISMQ()*, *S_TYPEISSEM()*, and *S_TYPEISSHM()* macros are unconditionally
12739 mandated.
- 12740 The Open Group Corrigendum U035/4 is applied. In the DESCRIPTION, the types **blksize_t**
12741 and **blkcnt_t** have been described.
- 12742 The following new requirements on POSIX implementations derive from alignment with the
12743 Single UNIX Specification:
- 12744 • The **dev_t**, **ino_t**, **mode_t**, **nlink_t**, **uid_t**, **gid_t**, **off_t**, and **time_t** types are mandated.
- 12745 *S_IFSOCK* and *S_ISSOCK* are added for sockets.
- 12746 The description of **stat** structure members is changed to reflect contents when file type is a
12747 symbolic link.
- 12748 The test macro *S_TYPEISTMO* is added for alignment with IEEE Std 1003.1j-2000.
- 12749 The **restrict** keyword is added to the prototypes for *lstat()* and *stat()*.
- 12750 The *lstat()* function is now mandatory.

12751 NAME

12752 sys/statvfs.h — VFS File System information structure

12753 SYNOPSIS

12754 XSI #include <sys/statvfs.h>

12755

12756 DESCRIPTION

12757 The <sys/statvfs.h> header shall define the **statvfs** structure that includes at least the following
12758 members:

- 12759 unsigned long f_bsize File system block size.
- 12760 unsigned long f_frsize Fundamental file system block size.
- 12761 fsblkcnt_t f_blocks Total number of blocks on file system in units of *f_frsize*.
- 12762 fsblkcnt_t f_bfree Total number of free blocks.
- 12763 fsblkcnt_t f_bavail Number of free blocks available to
12764 non-privileged process.
- 12765 fsfilcnt_t f_files Total number of file serial numbers.
- 12766 fsfilcnt_t f_ffree Total number of free file serial numbers.
- 12767 fsfilcnt_t f_favail Number of file serial numbers available to
12768 non-privileged process.
- 12769 unsigned long f_fsid File system ID.
- 12770 unsigned long f_flag Bit mask of *f_flag* values.
- 12771 unsigned long f_namemax Maximum filename length.

12772 The **fsblkcnt_t** and **fsfilcnt_t** types shall be defined as described in <sys/types.h>.

12773 The following flags for the *f_flag* member shall be defined:

- 12774 ST_RDONLY Read-only file system.
- 12775 ST_NOSUID Does not support setuid/setgid semantics.

12776 The following shall be declared as functions and may also be defined as macros. Function |
12777 prototypes shall be provided. |

```
12778 int statvfs(const char *restrict, struct statvfs *restrict);
12779 int fstatvfs(int, struct statvfs *);
```

12780 APPLICATION USAGE

12781 None.

12782 RATIONALE

12783 None.

12784 FUTURE DIRECTIONS

12785 None.

12786 SEE ALSO

12787 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *fstatvfs()*, *statvfs()*

12788 CHANGE HISTORY

12789 First released in Issue 4, Version 2.

12790 Issue 5

12791 The type of *f_blocks*, *f_bfree*, and *f_bavail* is changed from **unsigned long** to **fsblkcnt_t**; the type
12792 of *f_files*, *f_ffree*, and *f_favail* is changed from **unsigned long** to **fsfilcnt_t**.

12793 **Issue 6**

12794 The Open Group Corrigendum U035/5 is applied. In the DESCRIPTION, the types **fsblkcnt_t**
12795 and **fsfilcnt_t** have been described.

12796 The **restrict** keyword is added to the prototype for *statvfs()*.

12797 NAME

12798 sys/time.h — time types

12799 SYNOPSIS

12800 XSI #include <sys/time.h>

12801

12802 DESCRIPTION

12803 The <sys/time.h> header shall define the **timeval** structure that includes at least the following
12804 members:

12805 time_t tv_sec Seconds.

12806 suseconds_t tv_usec Microseconds.

12807 The <sys/time.h> header shall define the **itimerval** structure that includes at least the following
12808 members:

12809 struct timeval it_interval Timer interval.

12810 struct timeval it_value Current value.

12811 The **time_t** and **suseconds_t** types shall be defined as described in <sys/types.h>.

12812 The **fd_set** type shall be defined as described in <sys/select.h>.

12813 The <sys/time.h> header shall define the following values for the *which* argument of *getitimer()*
12814 and *setitimer()*:

12815 ITIMER_REAL Decrements in real time.

12816 ITIMER_VIRTUAL Decrements in process virtual time.

12817 ITIMER_PROF Decrements both in process virtual time and when the system is running
12818 on behalf of the process.

12819 The following shall be defined as described in <sys/select.h>:

12820 FD_CLR()

12821 FD_ISSET()

12822 FD_SET()

12823 FD_ZERO()

12824 FD_SETSIZE()

12825 The following shall be declared as functions and may also be defined as macros. Function
12826 prototypes shall be provided.

12827 int getitimer(int, struct itimerval *);

12828 int gettimeofday(struct timeval *restrict, void *restrict);

12829 int select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
12830 struct timeval *restrict);

12831 int setitimer(int, const struct itimerval *restrict,
12832 struct itimerval *restrict);

12833 int utimes(const char *, const struct timeval [2]); (**LEGACY**)

12834 Inclusion of the <sys/time.h> header may make visible all symbols from the <sys/select.h>
12835 header.

12836 **APPLICATION USAGE**

12837 None.

12838 **RATIONALE**

12839 None.

12840 **FUTURE DIRECTIONS**

12841 None.

12842 **SEE ALSO**12843 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *getitimer()*, *gettimeofday()*,12844 *select()*, *setitimer()*12845 **CHANGE HISTORY**

12846 First released in Issue 4, Version 2.

12847 **Issue 5**12848 The type of *tv_usec* is changed from **long** to **suseconds_t**.12849 **Issue 6**12850 The **restrict** keyword is added to the prototypes for *gettimeofday()*, *select()*, and *setitimer()*. |

12851 The note is added that inclusion of this header may also make symbols visible from |

12852 <sys/socket.h>. |

12853 The *utimes()* function is marked LEGACY. |

12854 **NAME**

12855 sys/timeb.h — additional definitions for date and time

12856 **SYNOPSIS**

12857 XSI #include <sys/timeb.h>

12858

12859 **DESCRIPTION**

12860 The <sys/timeb.h> header shall define the **timeb** structure that includes at least the following
12861 members:

12862	time_t	time	The seconds portion of the current time.
12863	unsigned short	millitm	The milliseconds portion of the current time.
12864	short	timezone	The local timezone in minutes west of Greenwich.
12865	short	dstflag	TRUE if Daylight Savings Time is in effect.

12866 The **time_t** type shall be defined as described in <sys/types.h>.

12867 The following shall be declared as a function and may also be defined as a macro. A function |
12868 prototype shall be provided. |

12869 int ftime(struct timeb *); (**LEGACY**)

12870 **APPLICATION USAGE**

12871 None.

12872 **RATIONALE**

12873 None.

12874 **FUTURE DIRECTIONS**

12875 None.

12876 **SEE ALSO**

12877 <sys/types.h>, <time.h>

12878 **CHANGE HISTORY**

12879 First released in Issue 4, Version 2.

12880 **Issue 6**

12881 The *ftime()* function is marked LEGACY.

12882 **NAME**

12883 sys/times.h — file access and modification times structure

12884 **SYNOPSIS**

12885 #include <sys/times.h>

12886 **DESCRIPTION**

12887 The <sys/times.h> header shall define the structure **tms**, which is returned by *times()* and
12888 includes at least the following members:

12889 clock_t tms_utime User CPU time.

12890 clock_t tms_stime System CPU time.

12891 clock_t tms_cutime User CPU time of terminated child processes.

12892 clock_t tms_cstime System CPU time of terminated child processes.

12893 The **clock_t** type shall be defined as described in <sys/types.h>.

12894 The following shall be declared as a function and may also be defined as a macro. A function |
12895 prototype shall be provided. |

12896 clock_t times(struct tms *);

12897 **APPLICATION USAGE**

12898 None.

12899 **RATIONALE**

12900 None.

12901 **FUTURE DIRECTIONS**

12902 None.

12903 **SEE ALSO**

12904 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *times()*

12905 **CHANGE HISTORY**

12906 First released in Issue 1. Derived from Issue 1 of the SVID.

12907 **NAME**

12908 sys/types.h — data types

12909 **SYNOPSIS**

12910 #include <sys/types.h>

12911 **DESCRIPTION**

12912 The <sys/types.h> header shall include definitions for at least the following types:

12913	blkcnt_t	Used for file block counts.
12914	blksize_t	Used for block sizes.
12915 XSI	clock_t	Used for system times in clock ticks or CLOCKS_PER_SEC; see
12916		<time.h>.
12917 TMR	clockid_t	Used for clock ID type in the clock and timer functions.
12918	dev_t	Used for device IDs.
12919 XSI	fsblkcnt_t	Used for file system block counts.
12920 XSI	fsfilcnt_t	Used for file system file counts.
12921	gid_t	Used for group IDs.
12922 XSI	id_t	Used as a general identifier; can be used to contain at least a pid_t ,
12923		uid_t , or gid_t .
12924	ino_t	Used for file serial numbers.
12925 XSI	key_t	Used for XSI interprocess communication.
12926	mode_t	Used for some file attributes.
12927	nlink_t	Used for link counts.
12928	off_t	Used for file sizes.
12929	pid_t	Used for process IDs and process group IDs.
12930 THR	pthread_attr_t	Used to identify a thread attribute object.
12931 BAR	pthread_barrier_t	Used to identify a barrier.
12932 BAR	pthread_barrierattr_t	Used to define a barrier attributes object.
12933 THR	pthread_cond_t	Used for condition variables.
12934 THR	pthread_condattr_t	Used to identify a condition attribute object.
12935 THR	pthread_key_t	Used for thread-specific data keys.
12936 THR	pthread_mutex_t	Used for mutexes.
12937 THR	pthread_mutexattr_t	Used to identify a mutex attribute object.
12938 THR	pthread_once_t	Used for dynamic package initialization.
12939 THR	pthread_rwlock_t	Used for read-write locks.
12940 THR	pthread_rwlockattr_t	Used for read-write lock attributes.
12941 SPI	pthread_spinlock_t	Used to identify a spin lock.
12942 THR	pthread_t	Used to identify a thread.

12943	size_t	Used for sizes of objects.
12944	ssize_t	Used for a count of bytes or an error indication.
12945 XSI	suseconds_t	Used for time in microseconds
12946	time_t	Used for time in seconds.
12947 TMR	timer_t	Used for timer ID returned by <i>timer_create()</i> .
12948	uid_t	Used for user IDs.
12949 XSI	useconds_t	Used for time in microseconds.
12950	All of the types shall be defined as arithmetic types of an appropriate length, with the following	
12951	exceptions:	
12952 XSI	key_t	
12953 THR	pthread_attr_t	
12954 BAR	pthread_barrier_t	
12955	pthread_barrierattr_t	
12956 THR	pthread_cond_t	
12957	pthread_condattr_t	
12958	pthread_key_t	
12959	pthread_mutex_t	
12960	pthread_mutexattr_t	
12961	pthread_once_t	
12962	pthread_rwlock_t	
12963	pthread_rwlockattr_t	
12964 SPI	pthread_spinlock_t	
12965 TRC	trace_attr_t	
12966	trace_event_id_t	
12967 TRC TEF	trace_event_set_t	
12968 TRC	trace_id_t	
12969		
12970	Additionally:	
12971	• mode_t shall be an integer type.	
12972	• nlink_t , uid_t , gid_t , and id_t shall be integer types.	
12973	• blkcnt_t and off_t shall be signed integer types.	
12974 XSI	• fsblkcnt_t , fsfilcnt_t , and ino_t shall be defined as unsigned integer types.	
12975	• size_t shall be an unsigned integer type.	
12976	• blksize_t , pid_t , and ssize_t shall be signed integer types.	
12977	• time_t and clock_t shall be integer or real-floating types.	
12978 XSI	The type ssize_t shall be capable of storing values at least in the range [-1, {SSIZE_MAX}]. The	
12979	type useconds_t shall be an unsigned integer type capable of storing values at least in the range	
12980	[0, 1 000 000]. The type suseconds_t shall be a signed integer type capable of storing values at	
12981	least in the range [-1, 1 000 000].	
12982	The implementation shall support one or more programming environments in which the widths	
12983	of blksize_t , pid_t , size_t , ssize_t , suseconds_t , and useconds_t are no greater than the width of	
12984	type long . The names of these programming environments can be obtained using the <i>confstr()</i>	
12985	function or the <i>getconf</i> utility.	

12986 There are no defined comparison or assignment operators for the following types: |

- 12987 THR **pthread_attr_t**
- 12988 BAR **pthread_barrier_t**
- 12989 **pthread_barrierattr_t**
- 12990 THR **pthread_cond_t**
- 12991 **pthread_condattr_t**
- 12992 **pthread_mutex_t**
- 12993 **pthread_mutexattr_t**
- 12994 **pthread_rwlock_t**
- 12995 **pthread_rwlockattr_t**
- 12996 SPI **pthread_spinlock_t**
- 12997 TRC **trace_attr_t**
- 12998

12999 **APPLICATION USAGE**

13000 None.

13001 **RATIONALE**

13002 None.

13003 **FUTURE DIRECTIONS**

13004 None.

13005 **SEE ALSO**

13006 <time.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *confstr()*, the Shell and Utilities |
13007 volume of IEEE Std 1003.1-200x, *getconf* |

13008 **CHANGE HISTORY**

13009 First released in Issue 1. Derived from Issue 1 of the SVID.

13010 **Issue 5**

13011 The **clockid_t** and **timer_t** types are defined for alignment with the POSIX Realtime Extension.

13012 The types **blkcnt_t**, **blksize_t**, **fsblkcnt_t**, **fsfilcnt_t**, and **suseconds_t** are added.

13013 Large File System extensions are added.

13014 Updated for alignment with the POSIX Threads Extension.

13015 **Issue 6**

13016 The **pthread_barrier_t**, **pthread_barrierattr_t**, and **pthread_spinlock_t** types are added for
13017 alignment with IEEE Std 1003.1j-2000.

13018 The margin code is changed from XSI to THR for the **pthread_rwlock_t** and
13019 **pthread_rwlockattr_t** types as Read-Write Locks have been absorbed into the POSIX Threads
13020 option. The threads types are now marked THR.

13021 **NAME**

13022 sys/uio.h — definitions for vector I/O operations

13023 **SYNOPSIS**13024 XSI

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

13025

13026 **DESCRIPTION**13027 The <sys/uio.h> header shall define the **iovec** structure that includes at least the following
13028 members:

13029 void *iov_base Base address of a memory region for input or output.

13030 size_t iov_len The size of the memory pointed to by *iov_base*.13031 The <sys/uio.h> header uses the **iovec** structure for scatter/gather I/O.13032 The **ssize_t** and **size_t** types shall be defined as described in <sys/types.h>.13033 The following shall be declared as functions and may also be defined as macros. Function |
13034 prototypes shall be provided. |

13035 ssize_t readv(int, const struct iovec *, int);

13036 ssize_t writev(int, const struct iovec *, int);

13037 **APPLICATION USAGE**13038 The implementation can put a limit on the number of scatter/gather elements which can be
13039 processed in one call. The symbol {IOV_MAX} defined in <limits.h> should always be used to
13040 learn about the limits instead of assuming a fixed value.13041 **RATIONALE**13042 Traditionally, the maximum number of scatter/gather elements the system can process in one
13043 call were described by the symbolic value {UIO_MAXIOV}. In IEEE Std 1003.1-200x this value
13044 was replaced by the constant {IOV_MAX} which can be found in <limits.h>.13045 **FUTURE DIRECTIONS**

13046 None.

13047 **SEE ALSO**13048 <limits.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *read()*, *write()*13049 **CHANGE HISTORY**

13050 First released in Issue 4, Version 2.

13051 **Issue 6**

13052 Text referring to scatter/gather I/O is added to the DESCRIPTION.

13053 **NAME**

13054 sys/un.h — definitions for UNIX domain sockets

13055 **SYNOPSIS**

13056 #include <sys/un.h>

13057 **DESCRIPTION**

13058 The <sys/un.h> header shall define the **sockaddr_un** structure that includes at least the
13059 following members:

13060 sa_family_t sun_family Address family.
13061 char sun_path[] Socket pathname.

13062 The **sockaddr_un** structure is used to store addresses for UNIX domain sockets. Values of this
13063 type shall be cast by applications to **struct sockaddr** for use with socket functions.

13064 The **sa_family_t** type shall be defined as described in <sys/socket.h>.

13065 **APPLICATION USAGE**

13066 The size of *sun_path* has intentionally been left undefined. This is because different
13067 implementations use different sizes. For example, BSD4.3 uses a size of 108, and BSD4.4 uses a
13068 size of 104. Since most implementations originate from BSD versions, the size is typically in the
13069 range 92 to 108.

13070 Applications should not assume a particular length for *sun_path* or assume that it can hold
13071 `_POSIX_PATH_MAX` characters (255).

13072 **RATIONALE**

13073 None.

13074 **FUTURE DIRECTIONS**

13075 None.

13076 **SEE ALSO**

13077 <sys/socket.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *bind()*, *socket()*,
13078 *socketpair()*

13079 **CHANGE HISTORY**

13080 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

13081 **NAME**

13082 sys/utsname.h — system name structure

13083 **SYNOPSIS**

13084 #include <sys/utsname.h>

13085 **DESCRIPTION**13086 The <sys/utsname.h> header shall define the structure **utsname** which shall include at least the
13087 following members:

13088 char sysname[] Name of this implementation of the operating system.
13089 char nodename[] Name of this node within an implementation-defined
13090 communications network.
13091 char release[] Current release level of this implementation.
13092 char version[] Current version level of this release.
13093 char machine[] Name of the hardware type on which the system is running.

13094 The character arrays are of unspecified size, but the data stored in them shall be terminated by a
13095 null byte.

13096 The following shall be declared as a function and may also be defined as a macro:

13097 int uname(struct utsname *);

13098 **APPLICATION USAGE**

13099 None.

13100 **RATIONALE**

13101 None.

13102 **FUTURE DIRECTIONS**

13103 None.

13104 **SEE ALSO**13105 The System Interfaces volume of IEEE Std 1003.1-200x, *uname()*13106 **CHANGE HISTORY**

13107 First released in Issue 1. Derived from Issue 1 of the SVID.

13108 NAME

13109 sys/wait.h — declarations for waiting

13110 SYNOPSIS

13111 #include <sys/wait.h>

13112 DESCRIPTION

13113 The <sys/wait.h> header shall define the following symbolic constants for use with *waitpid()*:

13114 WNOHANG Do not hang if no status is available; return immediately.

13115 WUNTRACED Report status of stopped child process.

13116 The <sys/wait.h> header shall define the following macros for analysis of process status values:

13117 WEXITSTATUS Return exit status.

13118 XSI WIFCONTINUED True if child has been continued

13119 WIFEXITED True if child exited normally.

13120 WIFSIGNALED True if child exited due to uncaught signal.

13121 WIFSTOPPED True if child is currently stopped.

13122 WSTOPSIG Return signal number that caused process to stop.

13123 WTERMSIG Return signal number that caused process to terminate.

13124 XSI The following symbolic constants shall be defined as possible values for the *options* argument to
13125 *waitid()*:

13126 WEXITED Wait for processes that have exited.

13127 WSTOPPED Status is returned for any child that has stopped upon receipt of a signal.

13128 WCONTINUED Status is returned for any child that was stopped and has been continued.

13129 WNOHANG Return immediately if there are no children to wait for.

13130 WNOWAIT Keep the process whose status is returned in *infop* in a waitable state.

13131 The type *idtype_t* shall be defined as an enumeration type whose possible values shall include
13132 at least the following:

13133 P_ALL

13134 P_PID

13135 P_PGID

13136

13137 The *id_t* and *pid_t* types shall be defined as described in <sys/types.h>.

13138 XSI The *siginfo_t* type shall be defined as described in <signal.h>.

13139 The *rusage* structure shall be defined as described in <sys/resource.h>.

13140 Inclusion of the <sys/wait.h> header may also make visible all symbols from <signal.h> and
13141 <sys/resource.h>.

13142 The following shall be declared as functions and may also be defined as macros. Function
13143 prototypes shall be provided.

13144 pid_t wait(int *);

13145 XSI int waitid(idtype_t, id_t, siginfo_t *, int);

13146 pid_t waitpid(pid_t, int *, int);

13147 **APPLICATION USAGE**

13148 None.

13149 **RATIONALE**

13150 None.

13151 **FUTURE DIRECTIONS**

13152 None.

13153 **SEE ALSO**13154 <signal.h>, <sys/resource.h>, <sys/types.h>, <sys/wait.h>, the System Interfaces volume of
13155 IEEE Std 1003.1-200x, *wait()*, *waitid()*13156 **CHANGE HISTORY**

13157 First released in Issue 3.

13158 Entry included for alignment with the POSIX.1-1988 standard.

13159 **Issue 6**13160 The *wait3()* function is removed.

13161 **NAME**13162 **syslog** — definitions for system error logging13163 **SYNOPSIS**

13164 XSI #include <syslog.h>

13165

13166 **DESCRIPTION**13167 The **<syslog.h>** header shall define the following symbolic constants, zero or more of which may
13168 be OR'ed together to form the *logopt* option of *openlog()*:

13169 LOG_PID Log the process ID with each message.

13170 LOG_CONS Log to the system console on error.

13171 LOG_NDELAY Connect to syslog daemon immediately.

13172 LOG_ODELAY Delay open until *syslog()* is called.

13173 LOG_NOWAIT Do not wait for child processes.

13174 The following symbolic constants shall be defined as possible values of the *facility* argument to
13175 *openlog()*:

13176 LOG_KERN Reserved for message generated by the system.

13177 LOG_USER Message generated by a process.

13178 LOG_MAIL Reserved for message generated by mail system.

13179 LOG_NEWS Reserved for message generated by news system.

13180 LOG_UUCP Reserved for message generated by UUCP system.

13181 LOG_DAEMON Reserved for message generated by system daemon.

13182 LOG_AUTH Reserved for message generated by authorization daemon.

13183 LOG_CRON Reserved for message generated by the clock daemon.

13184 LOG_LPR Reserved for message generated by printer system.

13185 LOG_LOCAL0 Reserved for local use.

13186 LOG_LOCAL1 Reserved for local use.

13187 LOG_LOCAL2 Reserved for local use.

13188 LOG_LOCAL3 Reserved for local use.

13189 LOG_LOCAL4 Reserved for local use.

13190 LOG_LOCAL5 Reserved for local use.

13191 LOG_LOCAL6 Reserved for local use.

13192 LOG_LOCAL7 Reserved for local use.

13193 The following shall be declared as macros for constructing the *maskpri* argument to *setlogmask()*.13194 The following macros expand to an expression of type **int** when the argument *pri* is an
13195 expression of type **int**:13196 LOG_MASK(*pri*) A mask for priority *pri*.13197 The following constants shall be defined as possible values for the *priority* argument of *syslog()*:

13198 LOG_EMERG A panic condition was reported to all processes.

13199 LOG_ALERT A condition that should be corrected immediately.

13200 LOG_CRIT A critical condition.

13201 LOG_ERR An error message.

13202 LOG_WARNING A warning message.

13203 LOG_NOTICE A condition requiring special handling.

13204 LOG_INFO A general information message.

13205 LOG_DEBUG A message useful for debugging programs.

13206 The following shall be declared as functions and may also be defined as macros. Function |
 13207 prototypes shall be provided. |

13208 void closelog(void);

13209 void openlog(const char *, int, int);

13210 int setlogmask(int);

13211 void syslog(int, const char *, ...);

13212 **APPLICATION USAGE**

13213 None.

13214 **RATIONALE**

13215 None.

13216 **FUTURE DIRECTIONS**

13217 None.

13218 **SEE ALSO**

13219 The System Interfaces volume of IEEE Std 1003.1-200x, *closelog()*

13220 **CHANGE HISTORY**

13221 First released in Issue 4, Version 2.

13222 **Issue 5**

13223 Moved to X/Open UNIX to BASE.

13224 **NAME**

13225 tar.h — extended tar definitions

13226 **SYNOPSIS**

13227 #include <tar.h>

13228 **DESCRIPTION**

13229 The <tar.h> header shall define header block definitions as follows.

13230 General definitions:

13231

13232

13233

13234

13235

13236

Name	Description	Value
TMAGIC	"ustar"	ustar plus null byte.
TMAGLEN	6	Length of the above.
TVERSION	"00"	00 without a null byte.
TVERSLEN	2	Length of the above.

13237

13237 *Typeflag* field definitions:

13238

13239

13240

13241

13242

13243

13244

13245

13246

13247

13248

Name	Description	Value
REGTYPE	'0'	Regular file.
AREGTYPE	'\0'	Regular file.
LNKTYPE	'1'	Link.
SYMTYPE	'2'	Symbolic link.
CHRTYPE	'3'	Character special.
BLKTYPE	'4'	Block special.
DIRTYPE	'5'	Directory.
FIFOTYPE	'6'	FIFO special.
CONTTYPE	'7'	Reserved.

13249

13249 *Mode* field bit definitions (octal):

13250

13251

13252

13253

13254 XSI

13255

13256

13257

13258

13259

13260

13261

13262

13263

Name	Description	Value
TSUID	04000	Set UID on execution.
TSGID	02000	Set GID on execution.
TSVTX	01000	On directories, restricted deletion flag.
TUREAD	00400	Read by owner.
TUWRITE	00200	Write by owner special.
TUEXEC	00100	Execute/search by owner.
TGREAD	00040	Read by group.
TGWRITE	00020	Write by group.
TGEXEC	00010	Execute/search by group.
TOREAD	00004	Read by other.
TOWRITE	00002	Write by other.
TOEXEC	00001	Execute/search by other.

13264 **APPLICATION USAGE**

13265 None.

13266 **RATIONALE**

13267 None.

13268 **FUTURE DIRECTIONS**

13269 None.

13270 **SEE ALSO**13271 The Shell and Utilities volume of IEEE Std 1003.1-200x, *pax*13272 **CHANGE HISTORY**

13273 First released in Issue 3. Derived from the entry in the POSIX.1-1988 standard.

13274 **Issue 6**13275 The **SEE ALSO** section now refers to *pax* since the Shell and Utilities volume of
13276 IEEE Std 1003.1-200x no longer contains the *tar* utility.

13277 **NAME**

13278 termios.h — define values for termios

13279 **SYNOPSIS**

13280 #include <termios.h>

13281 **DESCRIPTION**

13282 The <termios.h> header contains the definitions used by the terminal I/O interfaces (see
13283 Chapter 11 (on page 183) for the structures and names defined).

13284 **The termios Structure**

13285 The following data types shall be defined through **typedef**:

13286 **cc_t** Used for terminal special characters.

13287 **speed_t** Used for terminal baud rates.

13288 **tcflag_t** Used for terminal modes.

13289 The above types shall be all unsigned integer types.

13290 The implementation shall support one or more programming environments in which the widths |
13291 of **cc_t**, **speed_t**, and **tcflag_t** are no greater than the width of type **long**. The names of these |
13292 programming environments can be obtained using the *confstr()* function or the *getconf* utility. |

13293 The **termios** structure shall be defined, and shall include at least the following members: |

- 13294 tcflag_t c_iflag Input modes.
- 13295 tcflag_t c_oflag Output modes.
- 13296 tcflag_t c_cflag Control modes.
- 13297 tcflag_t c_lflag Local modes.
- 13298 cc_t c_cc[NCCS] Control characters.

13299 A definition shall be provided for:

13300 NCCS Size of the array *c_cc* for control characters.

13301 The following subscript names for the array *c_cc* shall be defined:

13302

13303

13304

13305

13306

13307

13308

13309

13310

13311

13312

13313

13314

13315

Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF		EOF character.
VEOL		EOL character.
VERASE		ERASE character.
VINTR	VINTR	INTR character.
VKILL		KILL character.
	VMIN	MIN value.
VQUIT	VQUIT	QUIT character.
VSTART	VSTART	START character.
VSTOP	VSTOP	STOP character.
VSUSP	VSUSP	SUSP character.
	VTIME	TIME value.

13316 The subscript values shall be unique, except that the VMIN and VTIME subscripts may have the
13317 same values as the VEOF and VEOL subscripts, respectively.

13318 The following flags shall be provided.

13319 **Input Modes**

13320 The *c_iflag* field describes the basic terminal input control:

- 13321 BRKINT Signal interrupt on break.
- 13322 ICRNL Map CR to NL on input.
- 13323 IGNBRK Ignore break condition.
- 13324 IGNCR Ignore CR.
- 13325 IGNPAR Ignore characters with parity errors.
- 13326 INLCR Map NL to CR on input.
- 13327 INPCK Enable input parity check.
- 13328 ISTRIP Strip character.
- 13329 XSI **IXANY** Enable any character to restart output.
- 13330 IXOFF Enable start/stop input control.
- 13331 IXON Enable start/stop output control.
- 13332 PARMRK Mark parity errors.

13333 **Output Modes**

13334 The *c_oflag* field specifies the system treatment of output:

- 13335 OPOST Post-process output.
- 13336 XSI **ONLCR** Map NL to CR-NL on output.
- 13337 **OCRNL** Map CR to NL on output.
- 13338 **ONOCR** No CR output at column 0.
- 13339 **ONLRET** NL performs CR function.
- 13340 **OFILL** Use fill characters for delay.
- 13341 **NLDLY** Select newline delays:
- 13342 **NL0** <newline> type 0.
- 13343 **NL1** <newline> type 1.
- 13344 **CRDLY** Select carriage-return delays:
- 13345 **CR0** Carriage-return delay type 0.
- 13346 **CR1** Carriage-return delay type 1.
- 13347 **CR2** Carriage-return delay type 2.
- 13348 **CR3** Carriage-return delay type 3.
- 13349 **TABDLY** Select horizontal-tab delays:
- 13350 **TAB0** Horizontal-tab delay type 0.
- 13351 **TAB1** Horizontal-tab delay type 1.
- 13352 **TAB2** Horizontal-tab delay type 2.

13353	TAB3	Expand tabs to spaces.
13354	BSDLY	Select backspace delays:
13355	BS0	Backspace-delay type 0.
13356	BS1	Backspace-delay type 1.
13357	VTDLY	Select vertical-tab delays:
13358	VT0	Vertical-tab delay type 0.
13359	VT1	Vertical-tab delay type 1.
13360	FFDLY	Select form-feed delays:
13361	FF0	Form-feed delay type 0.
13362	FF1	Form-feed delay type 1.

13363 **Baud Rate Selection**

13364 The input and output baud rates are stored in the **termios** structure. These are the valid values
13365 for objects of type **speed_t**. The following values shall be defined, but not all baud rates need be
13366 supported by the underlying hardware.

13367	B0	Hang up
13368	B50	50 baud
13369	B75	75 baud
13370	B110	110 baud
13371	B134	134.5 baud
13372	B150	150 baud
13373	B200	200 baud
13374	B300	300 baud
13375	B600	600 baud
13376	B1200	1200 baud
13377	B1800	1800 baud
13378	B2400	2400 baud
13379	B4800	4800 baud
13380	B9600	9600 baud
13381	B19200	19200 baud
13382	B38400	38400 baud

13383 **Control Modes**

13384 The *c_cflag* field describes the hardware control of the terminal; not all values specified are
 13385 required to be supported by the underlying hardware:

13386 CSIZE Character size:

13387 CS5 5 bits

13388 CS6 6 bits

13389 CS7 7 bits

13390 CS8 8 bits

13391 CSTOPB Send two stop bits, else one.

13392 CREAD Enable receiver.

13393 PARENB Parity enable.

13394 PARODD Odd parity, else even.

13395 HUPCL Hang up on last close.

13396 CLOCAL Ignore modem status lines.

13397 The implementation shall support the functionality associated with the symbols CS7, CS8, |
 13398 CSTOPB, PARODD, and PARENB. |

13399 **Local Modes**

13400 The *c_lflag* field of the argument structure is used to control various terminal functions:

13401 ECHO Enable echo.

13402 ECHOE Echo erase character as error-correcting backspace.

13403 ECHOK Echo KILL.

13404 ECHONL Echo NL.

13405 ICANON Canonical input (erase and kill processing).

13406 IEXTEN Enable extended input character processing.

13407 ISIG Enable signals.

13408 NOFLSH Disable flush after interrupt or quit.

13409 TOSTOP Send SIGTTOU for background output.

13410 **Attribute Selection**

13411 The following symbolic constants for use with *tcsetattr()* are defined:

13412 TCSANOW Change attributes immediately.

13413 TCSADRAIN Change attributes when output has drained.

13414 TCSAFLUSH Change attributes when output has drained; also flush pending input.

13415 **Line Control**

13416 The following symbolic constants for use with *tflush()* shall be defined:

13417 TCIFLUSH Flush pending input. Flush untransmitted output.

13418 TCIOFLUSH Flush both pending input and untransmitted output.

13419 TCOFLUSH Flush untransmitted output.

13420 The following symbolic constants for use with *tflow()* shall be defined:

13421 TCIOFF Transmit a STOP character, intended to suspend input data.

13422 TCION Transmit a START character, intended to restart input data.

13423 TCOOFF Suspend output.

13424 TCOON Restart output.

13425 The following shall be declared as functions and may also be defined as macros. Function
13426 prototypes shall be provided.

```

13427 speed_t cfgetispeed(const struct termios *);
13428 speed_t cfgetospeed(const struct termios *);
13429 int cfsetispeed(struct termios *, speed_t);
13430 int cfsetospeed(struct termios *, speed_t);
13431 int tcdrain(int);
13432 int tcflow(int, int);
13433 int tcflush(int, int);
13434 int tcgetattr(int, struct termios *);
13435 xsi pid_t tcgetsid(int);
13436 int tcsendbreak(int, int);
13437 int tcsetattr(int, int, const struct termios *);

```

13438 **APPLICATION USAGE**

13439 The following names are reserved for XSI-conformant systems to use as an extension to the
13440 above; therefore strictly conforming applications shall not use them:

13441	CBAUD	EXTB	VDSUSP
13442	DEFECHO	FLUSHO	VLNEXT
13443	ECHOCTL	LOBLK	VREPRINT
13444	ECHOKE	PENDIN	VSTATUS
13445	ECHOPRT	SWTCH	VWERASE
13446	EXTA	VDISCARD	

13447 **RATIONALE**

13448 None.

13449 **FUTURE DIRECTIONS**

13450 None.

13451 **SEE ALSO**

13452 The System Interfaces volume of IEEE Std 1003.1-200x, *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*,
13453 *cfsetospeed()*, *getconf()*, *tcdrain()*, *tcflow()*, *tcflush()*, *tcgetattr()*, *tcgetsid()*, *tcsendbreak()*, *tcsetattr()*,
13454 the Shell and Utilities volume of IEEE Std 1003.1-200x, *getconf*, Chapter 11 (on page 183)

13455 **CHANGE HISTORY**

13456 First released in Issue 3.

13457 Entry included for alignment with the ISO POSIX-1 standard.

13458 **Issue 6**

13459 The LEGACY symbols IUCLC, ULCUC, and XCASE are removed. |

13460 FIPS 151-2 requirements for the symbols CS7, CS8, CSTOPB, PARODD, and PARENB are |

13461 reaffirmed. |

13462 NAME

13463 tgmath.h — type-generic macros

13464 SYNOPSIS

13465 #include <tgmath.h>

13466 DESCRIPTION

13467 cx The functionality described on this reference page is aligned with the ISO C standard. Any
13468 conflict between the requirements described here and the ISO C standard is unintentional. This
13469 volume of IEEE Std 1003.1-200x defers to the ISO C standard.

13470 The <tgmath.h> header shall include the headers <math.h> and <complex.h> and shall define
13471 several type-generic macros.

13472 Of the functions contained within the <math.h> and <complex.h> headers without an *f* (**float**) or
13473 *l* (**long double**) suffix, several have one or more parameters whose corresponding real type is
13474 **double**. For each such function, except *modf()*, there shall be a corresponding type-generic
13475 macro. The parameters whose corresponding real type is **double** in the function synopsis are
13476 generic parameters. Use of the macro invokes a function whose corresponding real type and
13477 type domain are determined by the arguments for the generic parameters.

13478 Use of the macro invokes a function whose generic parameters have the corresponding real type
13479 determined as follows:

- 13480 • First, if any argument for generic parameters has type **long double**, the type determined is
13481 **long double**.
- 13482 • Otherwise, if any argument for generic parameters has type **double** or is of integer type, the
13483 type determined is **double**.
- 13484 • Otherwise, the type determined is **float**.

13485 For each unsuffixed function in the <math.h> header for which there is a function in the
13486 <complex.h> header with the same name except for a *c* prefix, the corresponding type-generic
13487 macro (for both functions) has the same name as the function in the <math.h> header. The
13488 corresponding type-generic macro for *fabs()* and *cabs()* is *fabs()*.

13489
13490
13491
13492
13493
13494
13495
13496
13497
13498
13499
13500
13501
13502
13503
13504
13505
13506
13507

<math.h> Function	<complex.h> Function	Type-Generic Macro
<i>acos()</i>	<i>cacos()</i>	<i>acos()</i>
<i>asin()</i>	<i>casin()</i>	<i>asin()</i>
<i>atan()</i>	<i>catan()</i>	<i>atan()</i>
<i>acosh()</i>	<i>cacosh()</i>	<i>acosh()</i>
<i>asinh()</i>	<i>casinh()</i>	<i>asinh()</i>
<i>atanh()</i>	<i>catanh()</i>	<i>atanh()</i>
<i>cos()</i>	<i>ccos()</i>	<i>cos()</i>
<i>sin()</i>	<i>csin()</i>	<i>sin()</i>
<i>tan()</i>	<i>ctan()</i>	<i>tan()</i>
<i>cosh()</i>	<i>ccosh()</i>	<i>cosh()</i>
<i>sinh()</i>	<i>csinh()</i>	<i>sinh()</i>
<i>tanh()</i>	<i>ctanh()</i>	<i>tanh()</i>
<i>exp()</i>	<i>cexp()</i>	<i>exp()</i>
<i>log()</i>	<i>clog()</i>	<i>log()</i>
<i>pow()</i>	<i>cpow()</i>	<i>pow()</i>
<i>sqrt()</i>	<i>csqrt()</i>	<i>sqrt()</i>
<i>fabs()</i>	<i>cabs()</i>	<i>fabs()</i>

13508 If at least one argument for a generic parameter is complex, then use of the macro invokes a
 13509 complex function; otherwise, use of the macro invokes a real function.

13510 For each unsuffixed function in the <math.h> header without a *c*-prefixed counterpart in the
 13511 <complex.h> header, the corresponding type-generic macro has the same name as the function.
 13512 These type-generic macros are:

13513	<i>atan2()</i>	<i>fma()</i>	<i>llround()</i>	<i>remainder()</i>	
13514	<i>cbrt()</i>	<i>fmax()</i>	<i>log10()</i>	<i>remquo()</i>	
13515	<i>ceil()</i>	<i>fmin()</i>	<i>log1p()</i>	<i>rint()</i>	
13516	<i>copysign()</i>	<i>fmod()</i>	<i>log2()</i>	<i>round()</i>	
13517	<i>erf()</i>	<i>frexp()</i>	<i>logb()</i>	<i>scalbn()</i>	
13518	<i>erfc()</i>	<i>hypot()</i>	<i>lrint()</i>	<i>scalbln()</i>	
13519	<i>exp2()</i>	<i>ilogb()</i>	<i>lround()</i>	<i>tgamma()</i>	
13520	<i>expm1()</i>	<i>ldexp()</i>	<i>nearbyint()</i>	<i>trunc()</i>	
13521	<i>fdim()</i>	<i>lgamma()</i>	<i>nextafter()</i>		
13522	<i>floor()</i>	<i>llrint()</i>	<i>nexttoward()</i>		

13523 If all arguments for generic parameters are real, then use of the macro invokes a real function;
 13524 otherwise, use of the macro results in undefined behavior.

13525 For each unsuffixed function in the <complex.h> header that is not a *c*-prefixed counterpart to a
 13526 function in the <math.h> header, the corresponding type-generic macro has the same name as
 13527 the function. These type-generic macros are:

13528	<i>carg()</i>	
13529	<i>cimag()</i>	
13530	<i>conj()</i>	
13531	<i>cproj()</i>	
13532	<i>creal()</i>	

13533 Use of the macro with any real or complex argument invokes a complex function.

13534 **APPLICATION USAGE**

13535 With the declarations:

```
13536 #include <tgmath.h>
13537 int n;
13538 float f;
13539 double d;
13540 long double ld;
13541 float complex fc;
13542 double complex dc;
13543 long double complex ldc;
```

13544 functions invoked by use of type-generic macros are shown in the following table:

Macro	Use Invokes
<i>exp(n)</i>	<i>exp(n)</i> , the function
<i>acosh(f)</i>	<i>acosh(f)</i>
<i>sin(d)</i>	<i>sin(d)</i> , the function
<i>atan(ld)</i>	<i>atanl(ld)</i>

13551

13552

13553

13554

13555

13556

13557

13558

13559

13560

13561

13562

13563

13564

13565

13566

13567

13568

13569

Macro	Use Invokes
<i>log(fc)</i>	<i>clogf(fc)</i>
<i>sqrtd(dc)</i>	<i>csqrtd(dc)</i>
<i>pow(ldc,f)</i>	<i>cpowl(ldc, f)</i>
<i>remainder(n,n)</i>	<i>remainder(n, n)</i> , the function
<i>nextafter(d,f)</i>	<i>nextafter(d, f)</i> , the function
<i>nexttoward(f,ld)</i>	<i>nexttowardf(f, ld)</i>
<i>copysign(n,ld)</i>	<i>copysignl(n, ld)</i>
<i>ceil(fc)</i>	Undefined behavior
<i>rint(dc)</i>	Undefined behavior
<i>fmax(ldc,ld)</i>	Undefined behavior
<i>carg(n)</i>	<i>carg(n)</i> , the function
<i>cproj(f)</i>	<i>cprojf(f)</i>
<i>creal(d)</i>	<i>creal(d)</i> , the function
<i>cimag(ld)</i>	<i>cimagl(ld)</i>
<i>cabs(fc)</i>	<i>cabsf(fc)</i>
<i>carg(dc)</i>	<i>carg(dc)</i> , the function
<i>cproj(ldc)</i>	<i>cprojl(ldc)</i>

13570 **RATIONALE**

13571

13572

13573

13574

13575

Type-generic macros allow calling a function whose type is determined by the argument type, as is the case for C operators such as '+' and '*'. For example, with a type-generic *cos()* macro, the expression *cos(float)x* will have type **float**. This feature enables writing more portably efficient code and alleviates need for awkward casting and suffixing in the process of porting or adjusting precision. Generic math functions are a widely appreciated feature of Fortran.

13576

13577

13578

13579

The only arguments that affect the type resolution are the arguments corresponding to the parameters that have type **double** in the synopsis. Hence the type of a type-generic call to *nexttoward()*, whose second parameter is **long double** in the synopsis, is determined solely by the type of the first argument.

13580

13581

13582

The term "type-generic" was chosen over the proposed alternatives of intrinsic and overloading. The term is more specific than intrinsic, which already is widely used with a more general meaning, and reflects a closer match to Fortran's generic functions than to C++ overloading.

13583

13584

The macros are placed in their own header in order not to silently break old programs that include the **<math.h>** header; for example, with:

13585

```
printf ("%e", sin(x))
```

13586

13587

*modf(double, double *)* is excluded because no way was seen to make it safe without complicating the type resolution.

13588

13589

13590

The implementation might, as an extension, endow appropriate ones of the macros that IEEE Std 1003.1-200x specifies only for real arguments with the ability to invoke the complex functions.

13591

13592

13593

IEEE Std 1003.1-200x does not prescribe any particular implementation mechanism for generic macros. It could be implemented simply with built-in macros. The generic macro for *sqrtd()*, for example, could be implemented with:

13594

13595

```
#undef sqrt
#define sqrt(x) __BUILTIN_GENERIC_sqrt(x)
```

13596

13597

Generic macros are designed for a useful level of consistency with C++ overloaded math functions.

13598 The great majority of existing C programs are expected to be unaffected when the <tgmath.h>
13599 header is included instead of the <math.h> or <complex.h> headers. Generic macros are similar
13600 to the ISO/IEC 9899:1999 standard library masking macros, though the semantic types of return
13601 values differ.

13602 The ability to overload on integer as well as floating types would have been useful for some
13603 functions; for example, *copysign()*. Overloading with different numbers of arguments would
13604 have allowed reusing names; for example, *remainder()* for *remquo()*. However, these facilities
13605 would have complicated the specification; and their natural consistent use, such as for a floating
13606 *abs()* or a two-argument *atan()*, would have introduced further inconsistencies with the
13607 ISO/IEC 9899:1999 standard for insufficient benefit.

13608 The ISO C standard in no way limits the implementation's options for efficiency, including
13609 inlining library functions.

13610 **FUTURE DIRECTIONS**

13611 None.

13612 **SEE ALSO**

13613 <math.h>, <complex.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *cabs()*, *fabs()*,
13614 *modf()*

13615 **CHANGE HISTORY**

13616 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

13617 **NAME**

13618 time.h — time types

13619 **SYNOPSIS**

13620 #include <time.h>

13621 **DESCRIPTION**

13622 **CX** Some of the functionality described on this reference page extends the ISO C standard.
 13623 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 13624 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these
 13625 symbols in this header.

13626 The **<time.h>** header shall declare the structure **tm**, which shall include at least the following
 13627 members:

13628	int	tm_sec	Seconds [0,60].
13629	int	tm_min	Minutes [0,59].
13630	int	tm_hour	Hour [0,23].
13631	int	tm_mday	Day of month [1,31].
13632	int	tm_mon	Month of year [0,11].
13633	int	tm_year	Years since 1900.
13634	int	tm_wday	Day of week [0,6] (Sunday =0).
13635	int	tm_yday	Day of year [0,365].
13636	int	tm_isdst	Daylight savings flag.

13637 The value of *tm_isdst* shall be positive if Daylight Saving Time is in effect, 0 if Daylight Saving
 13638 Time is not in effect, and negative if the information is not available.

13639 The **<time.h>** header shall define the following symbolic names:

13640	NULL	Null pointer constant.
13641	CLOCKS_PER_SEC	A number used to convert the value returned by the <i>clock()</i> function into 13642 seconds.

13643	TMR CPT	CLOCK_PROCESS_CPUTIME_ID
13644		The identifier of the CPU-time clock associated with the process making a 13645 <i>clock()</i> or <i>timer*()</i> function call.

13646	TMR TCT	CLOCK_THREAD_CPUTIME_ID
13647		The identifier of the CPU-time clock associated with the thread making a 13648 <i>clock()</i> or <i>timer*()</i> function call.

13649 **TMR** The **<time.h>** header shall declare the structure **timespec**, which has at least the following
 13650 members:

13651	time_t	tv_sec	Seconds.
13652	long	tv_nsec	Nanoseconds.

13653 The **<time.h>** header shall also declare the **itimerspec** structure, which has at least the following
 13654 members:

13655	struct timespec	it_interval	Timer period.
13656	struct timespec	it_value	Timer expiration.

13657 The following manifest constants shall be defined:

13658	CLOCK_REALTIME	The identifier of the system-wide realtime clock.
13659	TIMER_ABSTIME	Flag indicating time is absolute with respect to the clock associated with a 13660 timer.

13661 MON CLOCK_MONOTONIC
 13662 The identifier for the system-wide monotonic clock, which is defined as a
 13663 clock whose value cannot be set via *clock_settime()* and which cannot
 13664 have backward clock jumps. The maximum possible clock jump shall be
 13665 implementation-defined.

13666 TMR The *clock_t*, *size_t*, *time_t*, *clockid_t*, and *timer_t* types shall be defined as described in
 13667 <sys/types.h>.

13668 XSI Although the value of CLOCKS_PER_SEC is required to be 1 million on all XSI-conformant
 13669 systems, it may be variable on other systems, and it should not be assumed that
 13670 CLOCKS_PER_SEC is a compile-time constant.

13671 XSI The <time.h> header shall provide a declaration for *getdate_err*.

13672 The following shall be declared as functions and may also be defined as macros. Function
 13673 prototypes shall be provided.

```

13674 char      *asctime(const struct tm *);
13675 TSF char      *asctime_r(const struct tm *restrict, char *restrict);
13676 clock_t    clock(void);
13677 CPT int      clock_getcpuclockid(pid_t, clockid_t *);
13678 TMR int      clock_getres(clockid_t, struct timespec *);
13679 int      clock_gettime(clockid_t, struct timespec *);
13680 CS int      clock_nanosleep(clockid_t, int, const struct timespec *,
13681                          struct timespec *);
13682 TMR int      clock_settime(clockid_t, const struct timespec *);
13683 char      *ctime(const time_t *);
13684 TSF char      *ctime_r(const time_t *, char *);
13685 double     difftime(time_t, time_t);
13686 XSI struct tm *getdate(const char *);
13687 struct tm *gmtime(const time_t *);
13688 TSF struct tm *gmtime_r(const time_t *restrict, struct tm *restrict);
13689 struct tm *localtime(const time_t *);
13690 TSF struct tm *localtime_r(const time_t *restrict, struct tm *restrict);
13691 time_t     mktime(struct tm *);
13692 TMR int      nanosleep(const struct timespec *, struct timespec *);
13693 size_t     strftime(char *restrict, size_t, const char *restrict,
13694                  const struct tm *restrict);
13695 XSI char      *strptime(const char *restrict, const char *restrict,
13696                  struct tm *restrict);
13697 time_t     time(time_t *);
13698 TMR int      timer_create(clockid_t, struct sigevent *restrict,
13699                  timer_t *restrict);
13700 int      timer_delete(timer_t);
13701 int      timer_gettime(timer_t, struct itimerspec *);
13702 int      timer_getoverrun(timer_t);
13703 int      timer_settime(timer_t, int, const struct itimerspec *restrict,
13704                  struct itimerspec *restrict);
13705 CX void     tzset(void);
13706
13707 The following shall be declared as variables:
13708 XSI extern int     daylight;
13709 extern long    timezone;
```

13710 cx extern char *tzname[];
13711

13712 cx Inclusion of the **<time.h>** header may make visible all symbols from the **<signal.h>** header.

13713 APPLICATION USAGE

13714 The range [0,60] for *tm_sec* allows for the occasional leap second.

13715 *tm_year* is a signed value; therefore, years before 1900 may be represented.

13716 To obtain the number of clock ticks per second returned by the *times()* function, applications
13717 should call *sysconf(_SC_CLK_TCK)*.

13718 RATIONALE

13719 The range [0,60] seconds allows for positive or negative leap seconds. The formal definition of
13720 UTC does not permit double leap seconds, so all mention of double leap seconds has been
13721 removed, and the range shortened from the former [0,61] seconds seen in previous versions of
13722 POSIX.

13723 FUTURE DIRECTIONS

13724 None.

13725 SEE ALSO

13726 **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *asctime()*, *clock()*,
13727 *clock_getcpuclockid()*, *clock_getres()*, *clock_nanosleep()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*,
13728 *localtime()*, *mktime()*, *nanosleep()*, *strftime()*, *strptime()*, *sysconf()*, *time()*, *timer_create()*,
13729 *timer_delete()*, *timer_getoverrun()*, *tzname*, *tzset()*, *utime()*

13730 CHANGE HISTORY

13731 First released in Issue 1. Derived from Issue 1 of the SVID.

13732 Issue 5

13733 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
13734 Threads Extension.

13735 Issue 6

13736 The Open Group Corrigendum U035/6 is applied. In the DESCRIPTION, the types **clockid_t**
13737 and **timer_t** have been described.

13738 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 13739
 - The POSIX timer-related functions are now marked as part of the Timers option.

13740 The symbolic name CLK_TCK is removed. Application usage is added describing how its
13741 equivalent functionality can be obtained using *sysconf()*.

13742 The *clock_getcpuclockid()* function and manifest constants CLOCK_PROCESS_CPUTIME_ID and
13743 CLOCK_THREAD_CPUTIME_ID are added for alignment with IEEE Std 1003.1d-1999.

13744 The manifest constant CLOCK_MONOTONIC and the *clock_nanosleep()* function are added for
13745 alignment with IEEE Std 1003.1j-2000.

13746 The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- 13747
 - The range for seconds is changed from [0,61] to [0,60].
 - The **restrict** keyword is added to the prototypes for *asctime_r()*, *gmtime_r()*, *localtime_r()*,
13748 *strftime()*, *strptime()*, *timer_create()*, and *timer_settime()*.

13750 IEEE PASC Interpretation 1003.1 #84 is applied adding the statement that symbols from the
13751 **<signal.h>** header may be made visible when the **<time.h>** header is included.

13752

Extensions beyond the ISO C standard are now marked.

13753 NAME

13754 trace.h — tracing

13755 SYNOPSIS

13756 TRC #include <trace.h>

13757

13758 DESCRIPTION

13759 The <trace.h> header shall define the **posix_trace_event_info** structure that includes at least the
13760 following members:

13761	trace_event_id_t	posix_event_id
13762	pid_t	posix_pid
13763	void	*posix_prog_address
13764	int	posix_truncation_status
13765	struct timespec	posix_timestamp
13766 THR	pthread_t	posix_thread_id

13767

13768 The <trace.h> header shall define the **posix_trace_status_info** structure that includes at least the
13769 following members:

13770	int	posix_stream_status
13771	int	posix_stream_full_status
13772	int	posix_stream_overrun_status
13773 TRL	int	posix_stream_flush_status
13774	int	posix_stream_flush_error
13775	int	posix_log_overrun_status
13776	int	posix_log_full_status

13777

13778 The <trace.h> header shall define the following symbols:

13779	POSIX_TRACE_RUNNING
13780	POSIX_TRACE_SUSPENDED
13781	POSIX_TRACE_FULL
13782	POSIX_TRACE_NOT_FULL
13783	POSIX_TRACE_NO_OVERRUN
13784	POSIX_TRACE_OVERRUN
13785 TRL	POSIX_TRACE_FLUSHING
13786	POSIX_TRACE_NOT_FLUSHING
13787	POSIX_TRACE_NOT_TRUNCATED
13788	POSIX_TRACE_TRUNCATED_READ
13789	POSIX_TRACE_TRUNCATED_RECORD
13790 TRL	POSIX_TRACE_FLUSH
13791	POSIX_TRACE_LOOP
13792	POSIX_TRACE_UNTIL_FULL
13793 TRI	POSIX_TRACE_CLOSE_FOR_CHILD
13794	POSIX_TRACE_INHERITED
13795 TRL	POSIX_TRACE_APPEND
13796	POSIX_TRACE_LOOP
13797	POSIX_TRACE_UNTIL_FULL
13798 TEF	POSIX_TRACE_FILTER
13799 TRL	POSIX_TRACE_FLUSH_START
13800	POSIX_TRACE_FLUSH_STOP
13801	POSIX_TRACE_OVERFLOW

```

13802     POSIX_TRACE_RESUME
13803     POSIX_TRACE_START
13804     POSIX_TRACE_STOP
13805     POSIX_TRACE_UNNAMED_USER_EVENT

```

13806 The following types shall be defined as described in <sys/types.h>:

```

13807     trace_attr_t
13808     trace_id_t
13809     trace_event_id_t
13810 TEF trace_event_set_t
13811

```

13812 The following shall be declared as functions and may also be defined as macros. Function |
13813 prototypes shall be provided. |

```

13814     int  posix_trace_attr_destroy(trace_attr_t *);
13815     int  posix_trace_attr_getclockres(const trace_attr_t *,
13816         struct timespec *);
13817     int  posix_trace_attr_getcreatetime(const trace_attr_t *,
13818         struct timespec *);
13819     int  posix_trace_attr_getgenversion(const trace_attr_t *, char *);
13820 TRI  int  posix_trace_attr_getinherited(const trace_attr_t *restrict,
13821     int *restrict);
13822 TRL  int  posix_trace_attr_getlogfullpolicy(const trace_attr_t *restrict,
13823     int *restrict);
13824     int  posix_trace_attr_getlogsize(const trace_attr_t *restrict,
13825     size_t *restrict);
13826     int  posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict,
13827     size_t *restrict);
13828     int  posix_trace_attr_getmaxsystemeventsz(const trace_attr_t *restrict,
13829     size_t *restrict);
13830     int  posix_trace_attr_getmaxusereventsiz(const trace_attr_t *restrict,
13831     size_t, size_t *restrict);
13832     int  posix_trace_attr_getname(const trace_attr_t *, char *);
13833     int  posix_trace_attr_getstreamfullpolicy(const trace_attr_t *restrict,
13834     int *restrict);
13835     int  posix_trace_attr_getstreamsize(const trace_attr_t *restrict,
13836     size_t *restrict);
13837     int  posix_trace_attr_init(trace_attr_t *);
13838 TRI  int  posix_trace_attr_setinherited(trace_attr_t *, int);
13839 TRL  int  posix_trace_attr_setlogfullpolicy(trace_attr_t *, int);
13840     int  posix_trace_attr_setlogsize(trace_attr_t *, size_t);
13841     int  posix_trace_attr_setmaxdatasize(trace_attr_t *, size_t);
13842     int  posix_trace_attr_setname(trace_attr_t *, const char *);
13843     int  posix_trace_attr_setstreamsize(trace_attr_t *, size_t);
13844     int  posix_trace_attr_setstreamfullpolicy(trace_attr_t *, int);
13845     int  posix_trace_clear(trace_id_t);
13846 TRL  int  posix_trace_close(trace_id_t);
13847     int  posix_trace_create(pid_t, const trace_attr_t *restrict,
13848     trace_id_t *restrict);
13849 TRL  int  posix_trace_create_withlog(pid_t, const trace_attr_t *restrict,
13850     int, trace_id_t *restrict);
13851     void posix_trace_event(trace_event_id_t, const void *restrict, size_t);

```

```

13852     int  posix_trace_eventid_equal(trace_id_t, trace_event_id_t,
13853                                     trace_event_id_t);
13854     int  posix_trace_eventid_get_name(trace_id_t, trace_event_id_t, char *);
13855     int  posix_trace_eventid_open(const char *restrict,
13856                                     trace_event_id_t *restrict);
13857     int  posix_trace_eventtypelist_getnext_id(trace_id_t,
13858                                     trace_event_id_t *restrict, int *restrict);
13859     int  posix_trace_eventtypelist_rewind(trace_id_t);
13860 TEF    int  posix_trace_eventset_add(trace_event_id_t, trace_event_set_t *);
13861     int  posix_trace_eventset_del(trace_event_id_t, trace_event_set_t *);
13862     int  posix_trace_eventset_empty(trace_event_set_t *);
13863     int  posix_trace_eventset_fill(trace_event_set_t *, int);
13864     int  posix_trace_eventset_ismember(trace_event_id_t,
13865                                     const trace_event_set_t *restrict, int *restrict);
13866     int  posix_trace_flush(trace_id_t);
13867     int  posix_trace_get_attr(trace_id_t, trace_attr_t *);
13868 TEF    int  posix_trace_get_filter(trace_id_t, trace_event_set_t *);
13869     int  posix_trace_get_status(trace_id_t,
13870                                     struct posix_trace_status_info *);
13871     int  posix_trace_getnext_event(trace_id_t,
13872                                     struct posix_trace_event_info *restrict, void *restrict,
13873                                     size_t, size_t *restrict, int *restrict);
13874 TRL    int  posix_trace_open(int, trace_id_t *);
13875     int  posix_trace_rewind(trace_id_t);
13876 TEF    int  posix_trace_set_filter(trace_id_t, const trace_event_set_t *, int);
13877     int  posix_trace_shutdown(trace_id_t);
13878     int  posix_trace_start(trace_id_t);
13879     int  posix_trace_stop(trace_id_t);
13880 TMO    int  posix_trace_timedgetnext_event(trace_id_t,
13881                                     struct posix_trace_event_info *restrict, void *restrict,
13882                                     size_t, size_t *restrict, int *restrict,
13883                                     const struct timespec *restrict);
13884 TEF    int  posix_trace_trid_eventid_open(trace_id_t, const char *restrict,
13885                                     trace_event_id_t *restrict);
13886     int  posix_trace_trygetnext_event(trace_id_t,
13887                                     struct posix_trace_event_info *restrict, void *restrict, size_t,
13888                                     size_t *restrict, int *restrict);

```

13889 APPLICATION USAGE

13890 None.

13891 RATIONALE

13892 None.

13893 FUTURE DIRECTIONS

13894 None.

13895 SEE ALSO

13896 <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, Section 2.11, Tracing, the
13897 System Interfaces volume of IEEE Std 1003.1-200x, *posix_trace_attr_destroy()*,
13898 *posix_trace_attr_getclockres()*, *posix_trace_attr_getcreatetime()*, *posix_trace_attr_getgenversion()*,
13899 *posix_trace_attr_getinherited()*, *posix_trace_attr_getlogfullpolicy()*, *posix_trace_attr_getlogsize()*,
13900 *posix_trace_attr_getmaxdatasize()*, *posix_trace_attr_getmaxsystemeventsizesize()*,
13901 *posix_trace_attr_getmaxusereventsizesize()*, *posix_trace_attr_getname()*,

13902 *posix_trace_attr_getstreamfullpolicy()*, *posix_trace_attr_getstreamsize()*, *posix_trace_attr_init()*,
13903 *posix_trace_attr_setinherited()*, *posix_trace_attr_setlogfullpolicy()*, *posix_trace_attr_setlogsize()*,
13904 *posix_trace_attr_setmaxdatasize()*, *posix_trace_attr_setname()*, *posix_trace_attr_setstreamsize()*,
13905 *posix_trace_attr_setstreamfullpolicy()*, *posix_trace_clear()*, *posix_trace_close()*, *posix_trace_create()*,
13906 *posix_trace_create_withlog()*, *posix_trace_event()*, *posix_trace_eventid_equal()*,
13907 *posix_trace_eventid_get_name()*, *posix_trace_eventid_open()*, *posix_trace_eventtypelist_getnext_id()*,
13908 *posix_trace_eventtypelist_rewind()*, *posix_trace_eventset_add()*, *posix_trace_eventset_del()*,
13909 *posix_trace_eventset_empty()*, *posix_trace_eventset_fill()*, *posix_trace_eventset_ismember()*,
13910 *posix_trace_flush()*, *posix_trace_get_attr()*, *posix_trace_get_filter()*, *posix_trace_get_status()*,
13911 *posix_trace_getnext_event()*, *posix_trace_open()*, *posix_trace_rewind()*, *posix_trace_set_filter()*,
13912 *posix_trace_shutdown()*, *posix_trace_start()*, *posix_trace_stop()*, *posix_trace_timedgetnext_event()*,
13913 *posix_trace_trid_eventid_open()*, *posix_trace_trygetnext_event()*

13914 **CHANGE HISTORY**

13915 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

13916 **NAME**

13917 ucontext.h — user context

13918 **SYNOPSIS**13919 XSI `#include <ucontext.h>`

13920

13921 **DESCRIPTION**13922 The **<ucontext.h>** header shall define the **mcontext_t** type through **typedef**.13923 The **<ucontext.h>** header shall define the **ucontext_t** type as a structure that shall include at least
13924 the following members:

13925	<code>ucontext_t *uc_link</code>	Pointer to the context that is resumed 13926 when this context returns.
13927	<code>sigset_t uc_sigmask</code>	The set of signals that are blocked when this 13928 context is active.
13929	<code>stack_t uc_stack</code>	The stack used by this context.
13930	<code>mcontext_t uc_mcontext</code>	A machine-specific representation of the saved 13931 context.

13932 The types **sigset_t** and **stack_t** shall be defined as in **<signal.h>**.13933 The following shall be declared as functions and may also be defined as macros, Function |
13934 prototypes shall be provided. |

```
13935 int getcontext(ucontext_t *);  
13936 int setcontext(const ucontext_t *);  
13937 void makecontext(ucontext_t *, void (*)(void), int, ...);  
13938 int swapcontext(ucontext_t *restrict, const ucontext_t *restrict);
```

13939 **APPLICATION USAGE**

13940 None.

13941 **RATIONALE**

13942 None.

13943 **FUTURE DIRECTIONS**

13944 None.

13945 **SEE ALSO**13946 **<signal.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *getcontext()*, *makecontext()*,
13947 *sigaction()*, *sigprocmask()*, *sigaltstack()*13948 **CHANGE HISTORY**

13949 First released in Issue 4, Version 2.

13950 **NAME**

13951 ulimit.h — ulimit commands

13952 **SYNOPSIS**

13953 XSI #include <ulimit.h>

13954

13955 **DESCRIPTION**13956 The <ulimit.h> header shall define the symbolic constants used by the *ulimit()* function.

13957 Symbolic constants:

13958 UL_GETFSIZE Get maximum file size.

13959 UL_SETFSIZE Set maximum file size.

13960 The following shall be declared as a function and may also be defined as a macro. A function |
13961 prototype shall be provided. |

13962 long ulimit(int, ...);

13963 **APPLICATION USAGE**

13964 None.

13965 **RATIONALE**

13966 None.

13967 **FUTURE DIRECTIONS**

13968 None.

13969 **SEE ALSO**13970 The System Interfaces volume of IEEE Std 1003.1-200x, *ulimit()*13971 **CHANGE HISTORY**

13972 First released in Issue 3.

13973 **NAME**13974 `unistd.h` — standard symbolic constants and types13975 **SYNOPSIS**13976 `#include <unistd.h>`13977 **DESCRIPTION**

13978 The **<unistd.h>** header defines miscellaneous symbolic constants and types, and declares
13979 miscellaneous functions. The actual value of the constants are unspecified except as shown. The
13980 contents of this header are shown below.

13981 **Version Test Macros**

13982 The following symbolic constants shall be defined:

13983 `_POSIX_VERSION`

13984 Integer value indicating version of IEEE Std 1003.1 (C-language binding) to which the
13985 implementation conforms. For implementations conforming to IEEE Std 1003.1-200x, the
13986 value shall be 200xxxL.

13987 `_POSIX2_VERSION`

13988 Integer value indicating version of the Shell and Utilities volume of IEEE Std 1003.1 to
13989 which the implementation conforms. For implementations conforming to
13990 IEEE Std 1003.1-200x, the value shall be 200xxxL.

13991 The following symbolic constant shall be defined only if the implementation supports the XSI
13992 option; see Section 2.1.4 (on page 19).

13993 XSI `_XOPEN_VERSION`

13994 Integer value indicating version of the X/Open Portability Guide to which the
13995 implementation conforms. The value shall be 600.

13996 **Constants for Options and Option Groups**

13997 The following symbolic constants, if defined in **<unistd.h>**, shall have a value of `-1`, `0`, or greater,
13998 unless otherwise specified below. If these are undefined, the `fpathconf()`, `pathconf()`, or `sysconf()`
13999 functions can be used to determine whether the option is provided for a particular invocation of
14000 the application.

14001 If a symbolic constant is defined with the value `-1`, the option is not supported. Headers, data
14002 types, and function interfaces required only for the option need not be supplied. An application
14003 that attempts to use anything associated only with the option is considered to be requiring an
14004 extension.

14005 If a symbolic constant is defined with a value greater than zero, the option shall always be
14006 supported when the application is executed. All headers, data types, and functions shall be
14007 present and shall operate as specified.

14008 If a symbolic constant is defined with the value zero, all headers, data types, and functions shall
14009 be present. The application can check at runtime to see whether the option is supported by
14010 calling `fpathconf()`, `pathconf()`, or `sysconf()` with the indicated *name* parameter.

14011 Unless explicitly specified otherwise, the behavior of functions associated with an unsupported
14012 option is unspecified, and an application that uses such functions without first checking
14013 `fpathconf()`, `pathconf()`, or `sysconf()` is considered to be requiring an extension.

14014 For conformance requirements, refer to Chapter 2 (on page 15).

14015	ADV	_POSIX_ADVISORY_INFO	
14016		The implementation supports the Advisory Information option. If this symbol has a value	
14017		other than -1 or 0, it shall have the value 200xxxL.	
14018	AIO	_POSIX_ASYNCHRONOUS_IO	
14019		The implementation supports the Asynchronous Input and Output option. If this symbol	
14020		has a value other than -1 or 0, it shall have the value 200xxxL.	
14021	BAR	_POSIX_BARRIERS	
14022		The implementation supports the Barriers option. If this symbol has a value other than -1 or	
14023		0, it shall have the value 200xxxL.	
14024		_POSIX_CHOWN_RESTRICTED	
14025		The use of <i>chown()</i> and <i>fchown()</i> is restricted to a process with appropriate privileges, and	
14026		to changing the group ID of a file only to the effective group ID of the process or to one of	
14027		its supplementary group IDs. This symbol shall always be set to a value other than -1.	
14028	CS	_POSIX_CLOCK_SELECTION	
14029		The implementation supports the Clock Selection option. If this symbol has a value other	
14030		than -1 or 0, it shall have the value 200xxxL.	
14031	CPT	_POSIX_CPUTIME	
14032		The implementation supports the Process CPU-Time Clocks option. If this symbol has a	
14033		value other than -1 or 0, it shall have the value 200xxxL.	
14034	FSC	_POSIX_FSYNC	
14035		The implementation supports the File Synchronization option. If this symbol has a value	
14036		other than -1 or 0, it shall have the value 200xxxL.	
14037		_POSIX_JOB_CONTROL	
14038		The implementation supports job control. This symbol shall always be set to a value greater	
14039		than zero.	
14040	MF	_POSIX_MAPPED_FILES	
14041		The implementation supports the Memory Mapped Files option. If this symbol has a value	
14042		other than -1 or 0, it shall have the value 200xxxL.	
14043	ML	_POSIX_MEMLOCK	
14044		The implementation supports the Process Memory Locking option. If this symbol has a	
14045		value other than -1 or 0, it shall have the value 200xxxL.	
14046	MLR	_POSIX_MEMLOCK_RANGE	
14047		The implementation supports the Range Memory Locking option. If this symbol has a value	
14048		other than -1 or 0, it shall have the value 200xxxL.	
14049	MPR	_POSIX_MEMORY_PROTECTION	
14050		The implementation supports the Memory Protection option. If this symbol has a value	
14051		other than -1 or 0, it shall have the value 200xxxL.	
14052	MSG	_POSIX_MESSAGE_PASSING	
14053		The implementation supports the Message Passing option. If this symbol has a value other	
14054		than -1 or 0, it shall have the value 200xxxL.	
14055	MON	_POSIX_MONOTONIC_CLOCK	
14056		The implementation supports the Monotonic Clock option. If this symbol has a value other	
14057		than -1 or 0, it shall have the value 200xxxL.	
14058		_POSIX_NO_TRUNC	
14059		Pathname components longer than {NAME_MAX} generate an error. This symbol shall	

14060	always be set to a value other than -1.	
14061	_POSIX_PRIORITIZED_IO	
14062	The implementation supports the Prioritized Input and Output option. If this symbol has a	
14063	value other than -1 or 0, it shall have the value 200xxxL.	
14064	_POSIX_PRIORITY_SCHEDULING	
14065	The implementation supports the Process Scheduling option. If this symbol has a value	
14066	other than -1 or 0, it shall have the value 200xxxL.	
14067	_POSIX_RAW_SOCKETS	
14068	The implementation supports the Raw Sockets option. If this symbol has a value other than	
14069	-1 or 0, it shall have the value 200xxxL.	
14070	_POSIX_READER_WRITER_LOCKS	
14071	The implementation supports the Read-Write Locks option. This is always set to a value	
14072	greater than zero if the Threads option is supported. If this symbol has a value other than -1	
14073	or 0, it shall have the value 200xxxL.	
14074	_POSIX_REALTIME_SIGNALS	
14075	The implementation supports the Realtime Signals Extension option. If this symbol has a	
14076	value other than -1 or 0, it shall have the value 200xxxL.	
14077	_POSIX_REGEX	
14078	The implementation supports the Regular Expression Handling option. This symbol shall	
14079	always be set to a value greater than zero.	
14080	_POSIX_SAVED_IDS	
14081	Each process has a saved set-user-ID and a saved set-group-ID. This symbol shall always	
14082	be set to a value greater than zero.	
14083	_POSIX_SEMAPHORES	
14084	The implementation supports the Semaphores option. If this symbol has a value other than	
14085	-1 or 0, it shall have the value 200xxxL.	
14086	_POSIX_SHARED_MEMORY_OBJECTS	
14087	The implementation supports the Shared Memory Objects option. If this symbol has a value	
14088	other than -1 or 0, it shall have the value 200xxxL.	
14089	_POSIX_SHELL	
14090	The implementation supports the POSIX shell. This symbol shall always be set to a value	
14091	greater than zero.	
14092	_POSIX_SPAWN	
14093	The implementation supports the Spawn option. If this symbol has a value other than -1 or	
14094	0, it shall have the value 200xxxL.	
14095	_POSIX_SPIN_LOCKS	
14096	The implementation supports the Spin Locks option. If this symbol has a value other than	
14097	-1 or 0, it shall have the value 200xxxL.	
14098	_POSIX_SPORADIC_SERVER	
14099	The implementation supports the Process Sporadic Server option. If this symbol has a value	
14100	other than -1 or 0, it shall have the value 200xxxL.	
14101	_POSIX_SYNCHRONIZED_IO	
14102	The implementation supports the Synchronized Input and Output option. If this symbol	
14103	has a value other than -1 or 0, it shall have the value 200xxxL.	

14104	TSA	_POSIX_THREAD_ATTR_STACKADDR	
14105		The implementation supports the Thread Stack Address Attribute option. If this symbol	
14106		has a value other than -1 or 0, it shall have the value 200xxxL.	
14107	TSS	_POSIX_THREAD_ATTR_STACKSIZE	
14108		The implementation supports the Thread Stack Address Size option. If this symbol has a	
14109		value other than -1 or 0, it shall have the value 200xxxL.	
14110	TCT	_POSIX_THREAD_CPUTIME	
14111		The implementation supports the Thread CPU-Time Clocks option. If this symbol has a	
14112		value other than -1 or 0, it shall have the value 200xxxL.	
14113	TPI	_POSIX_THREAD_PRIO_INHERIT	
14114		The implementation supports the Thread Priority Inheritance option. If this symbol has a	
14115		value other than -1 or 0, it shall have the value 200xxxL.	
14116	TPP	_POSIX_THREAD_PRIO_PROTECT	
14117		The implementation supports the Thread Priority Protection option. If this symbol has a	
14118		value other than -1 or 0, it shall have the value 200xxxL.	
14119	TPS	_POSIX_THREAD_PRIORITY_SCHEDULING	
14120		The implementation supports the Thread Execution Scheduling option. If this symbol has a	
14121		value other than -1 or 0, it shall have the value 200xxxL.	
14122	TSH	_POSIX_THREAD_PROCESS_SHARED	
14123		The implementation supports the Thread Process-Shared Synchronization option. If this	
14124		symbol has a value other than -1 or 0, it shall have the value 200xxxL.	
14125	TSF	_POSIX_THREAD_SAFE_FUNCTIONS	
14126		The implementation supports the Thread-Safe Functions option. If this symbol has a value	
14127		other than -1 or 0, it shall have the value 200xxxL.	
14128	TSP	_POSIX_THREAD_SPORADIC_SERVER	
14129		The implementation supports the Thread Sporadic Server option. If this symbol has a value	
14130		other than -1 or 0, it shall have the value 200xxxL.	
14131	THR	_POSIX_THREADS	
14132		The implementation supports the Threads option. If this symbol has a value other than -1	
14133		or 0, it shall have the value 200xxxL.	
14134	TMO	_POSIX_TIMEOUTS	
14135		The implementation supports the Timeouts option. If this symbol has a value other than -1	
14136		or 0, it shall have the value 200xxxL.	
14137	TMR	_POSIX_TIMERS	
14138		The implementation supports the Timers option. If this symbol has a value other than -1 or	
14139		0, it shall have the value 200xxxL.	
14140	TRC	_POSIX_TRACE	
14141		The implementation supports the Trace option. If this symbol has a value other than -1 or 0,	
14142		it shall have the value 200xxxL.	
14143	TEF	_POSIX_TRACE_EVENT_FILTER	
14144		The implementation supports the Trace Event Filter option. If this symbol has a value other	
14145		than -1 or 0, it shall have the value 200xxxL.	
14146	TRI	_POSIX_TRACE_INHERIT	
14147		The implementation supports the Trace Inherit option. If this symbol has a value other than	
14148		-1 or 0, it shall have the value 200xxxL.	

14149	TRL	_POSIX_TRACE_LOG	
14150		The implementation supports the Trace Log option. If this symbol has a value other than -1	
14151		or 0, it shall have the value 200xxxL.	
14152	TYM	_POSIX_TYPED_MEMORY_OBJECTS	
14153		The implementation supports the Typed Memory Objects option. If this symbol has a value	
14154		other than -1 or 0, it shall have the value 200xxxL.	
14155		_POSIX_VDISABLE	
14156		This symbol shall be defined to be the value of a character that shall disable terminal special	
14157		character handling as described in <termios.h>. This symbol shall always be set to a value	
14158		other than -1.	
14159		_POSIX2_C_BIND	
14160		The implementation supports the C-Language Binding option. This symbol shall always	
14161		have the value 200xxxL.	
14162	CD	_POSIX2_C_DEV	
14163		The implementation supports the C-Language Development Utilities option. If this symbol	
14164		has a value other than -1 or 0, it shall have the value 200xxxL.	
14165		_POSIX2_CHAR_TERM	
14166		The implementation supports at least one terminal type.	
14167	FD	_POSIX2_FORT_DEV	
14168		The implementation supports the FORTRAN Development Utilities option. If this symbol	
14169		has a value other than -1 or 0, it shall have the value 200xxxL.	
14170	FR	_POSIX2_FORT_RUN	
14171		The implementation supports the FORTRAN Runtime Utilities option. If this symbol has a	
14172		value other than -1 or 0, it shall have the value 200xxxL.	
14173		_POSIX2_LOCALEDEF	
14174		The implementation supports the creation of locales by the <i>localedef</i> utility. If this symbol	
14175		has a value other than -1 or 0, it shall have the value 200xxxL.	
14176	BE	_POSIX2_PBS	
14177		The implementation supports the Batch Environment Services and Utilities option. If this	
14178		symbol has a value other than -1 or 0, it shall have the value 200xxxL.	
14179	BE	_POSIX2_PBS_ACCOUNTING	
14180		The implementation supports the Batch Accounting option. If this symbol has a value other	
14181		than -1 or 0, it shall have the value 200xxxL.	
14182	BE	_POSIX2_PBS_CHECKPOINT	
14183		The implementation supports the Batch Checkpoint/Restart option. If this symbol has a	
14184		value other than -1 or 0, it shall have the value 200xxxL.	
14185	BE	_POSIX2_PBS_LOCATE	
14186		The implementation supports the Locate Batch Job Request option. If this symbol has a	
14187		value other than -1 or 0, it shall have the value 200xxxL.	
14188	BE	_POSIX2_PBS_MESSAGE	
14189		The implementation supports the Batch Job Message Request option. If this symbol has a	
14190		value other than -1 or 0, it shall have the value 200xxxL.	
14191	BE	_POSIX2_PBS_TRACK	
14192		The implementation supports the Track Batch Job Request option. If this symbol has a value	
14193		other than -1 or 0, it shall have the value 200xxxL.	

14194	SD	_POSIX2_SW_DEV	
14195			The implementation supports the Software Development Utilities option. If this symbol has
14196			a value other than <code>-1</code> or <code>0</code> , it shall have the value <code>200xxxL</code> .
14197	UP	_POSIX2_UPE	
14198			The implementation supports the User Portability Utilities option. If this symbol has a value
14199			other than <code>-1</code> or <code>0</code> , it shall have the value <code>200xxxL</code> .
14200		_V6_ILP32_OFF32	
14201			The implementation provides a C-language compilation environment with 32-bit int , long ,
14202			pointer , and off_t types.
14203		_V6_ILP32_OFFBIG	
14204			The implementation provides a C-language compilation environment with 32-bit int , long ,
14205			and pointer types and an off_t type using at least 64 bits.
14206		_V6_LP64_OFF64	
14207			The implementation provides a C-language compilation environment with 32-bit int and
14208			64-bit long , pointer , and off_t types.
14209		_V6_LPBIG_OFFBIG	
14210			The implementation provides a C-language compilation environment with an int type
14211			using at least 32 bits and long , pointer , and off_t types using at least 64 bits.
14212	XSI	_XBS5_ILP32_OFF32 (LEGACY)	
14213			The implementation provides a C-language compilation environment with 32-bit int , long ,
14214			pointer , and off_t types.
14215	XSI	_XBS5_ILP32_OFFBIG (LEGACY)	
14216			The implementation provides a C-language compilation environment with 32-bit int , long ,
14217			and pointer types and an off_t type using at least 64 bits.
14218	XSI	_XBS5_LP64_OFF64 (LEGACY)	
14219			The implementation provides a C-language compilation environment with 32-bit int and
14220			64-bit long , pointer , and off_t types.
14221	XSI	_XBS5_LPBIG_OFFBIG (LEGACY)	
14222			The implementation provides a C-language compilation environment with an int type
14223			using at least 32 bits and long , pointer , and off_t types using at least 64 bits.
14224	XSI	_XOPEN_CRYPT	
14225			The implementation supports the X/Open Encryption Option Group.
14226		_XOPEN_ENH_I18N	
14227			The implementation supports the Issue 4, Version 2 Enhanced Internationalization Option
14228			Group. This symbol shall always be set to a value other than <code>-1</code> .
14229		_XOPEN_LEGACY	
14230			The implementation supports the Legacy Option Group.
14231		_XOPEN_REALTIME	
14232			The implementation supports the X/Open Realtime Option Group.
14233		_XOPEN_REALTIME_THREADS	
14234			The implementation supports the X/Open Realtime Threads Option Group.
14235		_XOPEN_SHM	
14236			The implementation supports the Issue 4, Version 2 Shared Memory Option Group. This
14237			symbol shall always be set to a value other than <code>-1</code> .

14238 **XOPEN_STREAMS**
 14239 The implementation supports the XSI STREAMS Option Group.

14240 XSI **XOPEN_UNIX**
 14241 The implementation supports the XSI extension.

14242 **Execution-Time Symbolic Constants**

14243 If any of the following constants are not defined in the <unistd.h> header, the value shall vary
 14244 depending on the file to which it is applied.

14245 If any of the following constants are defined to have value -1 in the <unistd.h> header, the
 14246 implementation shall not provide the option on any file; if any are defined to have a value other
 14247 than -1 in the <unistd.h> header, the implementation shall provide the option on all applicable
 14248 files.

14249 All of the following constants, whether defined in <unistd.h> or not, may be queried with
 14250 respect to a specific file using the *pathconf()* or *fpathconf()* functions:

14251 **_POSIX_ASYNC_IO**
 14252 Asynchronous input or output operations may be performed for the associated file.

14253 **_POSIX_PRIO_IO**
 14254 Prioritized input or output operations may be performed for the associated file.

14255 **_POSIX_SYNC_IO**
 14256 Synchronized input or output operations may be performed for the associated file.

14257 **Constants for Functions**

14258 The following symbolic constant shall be defined:

14259 **NULL** Null pointer

14260 The following symbolic constants shall be defined for the *access()* function:

14261 **F_OK** Test for existence of file.

14262 **R_OK** Test for read permission.

14263 **W_OK** Test for write permission.

14264 **X_OK** Test for execute (search) permission.

14265 The constants **F_OK**, **R_OK**, **W_OK**, and **X_OK** and the expressions *R_OK | W_OK*, *R_OK | X_OK*,
 14266 and *R_OK | W_OK | X_OK* shall all have distinct values.

14267 The following symbolic constants shall be defined for the *confstr()* function:

14268 **_CS_POSIX_PATH**
 14269 This is the value for the *PATH* environment variable that finds all standard utilities.

14270 **_CS_POSIX_V6_ILP32_OFF32_CFLAGS**
 14271 If *sysconf(_SC_V6_ILP32_OFF32)* returns -1, the meaning of this value is unspecified.
 14272 Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to
 14273 build an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t**
 14274 types.

14275 **_CS_POSIX_V6_ILP32_OFF32_LDFLAGS**
 14276 If *sysconf(_SC_V6_ILP32_OFF32)* returns -1, the meaning of this value is unspecified.
 14277 Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build
 14278 an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

14279 `_CS_POSIX_V6_ILP32_OFF32_LIBS` |
 14280 If `sysconf(_SC_V6_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.
 14281 Otherwise, this value is the set of libraries to be given to the `cc` and `c99` utilities to build an
 14282 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

14283 `_CS_POSIX_V6_ILP32_OFF32_LINTFLAGS` |
 14284 If `sysconf(_SC_V6_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.
 14285 Otherwise, this value is the set of options to be given to the `lint` utility to check application
 14286 source using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

14287 `_CS_POSIX_V6_ILP32_OFFBIG_CFLAGS` |
 14288 If `sysconf(_SC_V6_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
 14289 Otherwise, this value is the set of initial options to be given to the `cc` and `c99` utilities to
 14290 build an application using a programming model with 32-bit **int**, **long**, and **pointer** types,
 14291 and an **off_t** type using at least 64 bits.

14292 `_CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS` |
 14293 If `sysconf(_SC_V6_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
 14294 Otherwise, this value is the set of final options to be given to the `cc` and `c99` utilities to build
 14295 an application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
 14296 **off_t** type using at least 64 bits.

14297 `_CS_POSIX_V6_ILP32_OFFBIG_LIBS` |
 14298 If `sysconf(_SC_V6_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
 14299 Otherwise, this value is the set of libraries to be given to the `cc` and `c99` utilities to build an
 14300 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
 14301 **off_t** type using at least 64 bits.

14302 `_CS_POSIX_V6_ILP32_OFFBIG_LINTFLAGS` |
 14303 If `sysconf(_SC_V6_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
 14304 Otherwise, this value is the set of options to be given to the `lint` utility to check an
 14305 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
 14306 **off_t** type using at least 64 bits.

14307 `_CS_POSIX_V6_LP64_OFF64_CFLAGS` |
 14308 If `sysconf(_SC_V6_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.
 14309 Otherwise, this value is the set of initial options to be given to the `cc` and `c99` utilities to
 14310 build an application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t**
 14311 types.

14312 `_CS_POSIX_V6_LP64_OFF64_LDFLAGS` |
 14313 If `sysconf(_SC_V6_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.
 14314 Otherwise, this value is the set of final options to be given to the `cc` and `c99` utilities to build
 14315 an application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

14316 `_CS_POSIX_V6_LP64_OFF64_LIBS` |
 14317 If `sysconf(_SC_V6_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.
 14318 Otherwise, this value is the set of libraries to be given to the `cc` and `c99` utilities to build an
 14319 application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

14320 `_CS_POSIX_V6_LP64_OFF64_LINTFLAGS` |
 14321 If `sysconf(_SC_V6_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.
 14322 Otherwise, this value is the set of options to be given to the `lint` utility to check application
 14323 source using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

14324 `_CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS` |
 14325 If `sysconf(_SC_V6_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.

14326 Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to
 14327 build an application using a programming model with an **int** type using at least 32 bits and
 14328 **long**, **pointer**, and **off_t** types using at least 64 bits.

14329 **_CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS**
 14330 If *sysconf*(*_SC_V6_LPBIG_OFFBIG*) returns -1, the meaning of this value is unspecified.
 14331 Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build
 14332 an application using a programming model with an **int** type using at least 32 bits and **long**,
 14333 **pointer**, and **off_t** types using at least 64 bits.

14334 **_CS_POSIX_V6_LPBIG_OFFBIG_LIBS**
 14335 If *sysconf*(*_SC_V6_LPBIG_OFFBIG*) returns -1, the meaning of this value is unspecified.
 14336 Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an
 14337 application using a programming model with an **int** type using at least 32 bits and **long**,
 14338 **pointer**, and **off_t** types using at least 64 bits.

14339 **_CS_POSIX_V6_LPBIG_OFFBIG_LINTFLAGS**
 14340 If *sysconf*(*_SC_V6_LPBIG_OFFBIG*) returns -1, the meaning of this value is unspecified.
 14341 Otherwise, this value is the set of options to be given to the *lint* utility to check application
 14342 source using a programming model with an **int** type using at least 32 bits and **long**, **pointer**,
 14343 and **off_t** types using at least 64 bits.

14344 **_CS_V6_WIDTH_RESTRICTED_ENVS**
 14345 This value is a <newline>-separated list of names of programming environments supported
 14346 by the implementation in which the widths of the **blksize_t**, **cc_t**, **mode_t**, **nfds_t**, **pid_t**,
 14347 **ptrdiff_t**, **size_t**, **speed_t**, **ssize_t**, **suseconds_t**, **tcflag_t**, **useconds_t**, **wchar_t**, and **wint_t**
 14348 types are no greater than the width of type **long**.

14349 XSI **_CS_XBS5_ILP32_OFF32_CFLAGS (LEGACY)**
 14350 If *sysconf*(*_SC_XBS5_ILP32_OFF32*) returns -1, the meaning of this value is unspecified.
 14351 Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to
 14352 build an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t**
 14353 types.

14354 XSI **_CS_XBS5_ILP32_OFF32_LDFLAGS (LEGACY)**
 14355 If *sysconf*(*_SC_XBS5_ILP32_OFF32*) returns -1, the meaning of this value is unspecified.
 14356 Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build
 14357 an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

14358 XSI **_CS_XBS5_ILP32_OFF32_LIBS (LEGACY)**
 14359 If *sysconf*(*_SC_XBS5_ILP32_OFF32*) returns -1, the meaning of this value is unspecified.
 14360 Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an
 14361 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

14362 XSI **_CS_XBS5_ILP32_OFF32_LINTFLAGS (LEGACY)**
 14363 If *sysconf*(*_SC_XBS5_ILP32_OFF32*) returns -1, the meaning of this value is unspecified.
 14364 Otherwise, this value is the set of options to be given to the *lint* utility to check application
 14365 source using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

14366 XSI **_CS_XBS5_ILP32_OFFBIG_CFLAGS (LEGACY)**
 14367 If *sysconf*(*_SC_XBS5_ILP32_OFFBIG*) returns -1, the meaning of this value is unspecified.
 14368 Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to
 14369 build an application using a programming model with 32-bit **int**, **long**, and **pointer** types,
 14370 and an **off_t** type using at least 64 bits.

14371 XSI **_CS_XBS5_ILP32_OFFBIG_LDFLAGS (LEGACY)**
 14372 If *sysconf*(*_SC_XBS5_ILP32_OFFBIG*) returns -1, the meaning of this value is unspecified.

14373		Otherwise, this value is the set of final options to be given to the <i>cc</i> and <i>c99</i> utilities to build an application using a programming model with 32-bit int , long , and pointer types, and an off_t type using at least 64 bits.
14374		
14375		
14376	XSI	_CS_XBS5_ILP32_OFFBIG_LIBS (LEGACY)
14377		If <i>sysconf</i> (<i>_SC_XBS5_ILP32_OFFBIG</i>) returns -1 , the meaning of this value is unspecified.
14378		Otherwise, this value is the set of libraries to be given to the <i>cc</i> and <i>c99</i> utilities to build an application using a programming model with 32-bit int , long , and pointer types, and an off_t type using at least 64 bits.
14379		
14380		
14381	XSI	_CS_XBS5_ILP32_OFFBIG_LINTFLAGS (LEGACY)
14382		If <i>sysconf</i> (<i>_SC_XBS5_ILP32_OFFBIG</i>) returns -1 , the meaning of this value is unspecified.
14383		Otherwise, this value is the set of options to be given to the <i>lint</i> utility to check an application using a programming model with 32-bit int , long , and pointer types, and an off_t type using at least 64 bits.
14384		
14385		
14386	XSI	_CS_XBS5_LP64_OFF64_CFLAGS (LEGACY)
14387		If <i>sysconf</i> (<i>_SC_XBS5_LP64_OFF64</i>) returns -1 , the meaning of this value is unspecified.
14388		Otherwise, this value is the set of initial options to be given to the <i>cc</i> and <i>c99</i> utilities to build an application using a programming model with 64-bit int , long , pointer , and off_t types.
14389		
14390		
14391	XSI	_CS_XBS5_LP64_OFF64_LDFLAGS (LEGACY)
14392		If <i>sysconf</i> (<i>_SC_XBS5_LP64_OFF64</i>) returns -1 , the meaning of this value is unspecified.
14393		Otherwise, this value is the set of final options to be given to the <i>cc</i> and <i>c99</i> utilities to build an application using a programming model with 64-bit int , long , pointer , and off_t types.
14394		
14395	XSI	_CS_XBS5_LP64_OFF64_LIBS (LEGACY)
14396		If <i>sysconf</i> (<i>_SC_XBS5_LP64_OFF64</i>) returns -1 , the meaning of this value is unspecified.
14397		Otherwise, this value is the set of libraries to be given to the <i>cc</i> and <i>c99</i> utilities to build an application using a programming model with 64-bit int , long , pointer , and off_t types.
14398		
14399	XSI	_CS_XBS5_LP64_OFF64_LINTFLAGS (LEGACY)
14400		If <i>sysconf</i> (<i>_SC_XBS5_LP64_OFF64</i>) returns -1 , the meaning of this value is unspecified.
14401		Otherwise, this value is the set of options to be given to the <i>lint</i> utility to check application source using a programming model with 64-bit int , long , pointer , and off_t types.
14402		
14403	XSI	_CS_XBS5_LPBIG_OFFBIG_CFLAGS (LEGACY)
14404		If <i>sysconf</i> (<i>_SC_XBS5_LPBIG_OFFBIG</i>) returns -1 , the meaning of this value is unspecified.
14405		Otherwise, this value is the set of initial options to be given to the <i>cc</i> and <i>c99</i> utilities to build an application using a programming model with an int type using at least 32 bits and long , pointer , and off_t types using at least 64 bits.
14406		
14407		
14408	XSI	_CS_XBS5_LPBIG_OFFBIG_LDFLAGS (LEGACY)
14409		If <i>sysconf</i> (<i>_SC_XBS5_LPBIG_OFFBIG</i>) returns -1 , the meaning of this value is unspecified.
14410		Otherwise, this value is the set of final options to be given to the <i>cc</i> and <i>c99</i> utilities to build an application using a programming model with an int type using at least 32 bits and long , pointer , and off_t types using at least 64 bits.
14411		
14412		
14413	XSI	_CS_XBS5_LPBIG_OFFBIG_LIBS (LEGACY)
14414		If <i>sysconf</i> (<i>_SC_XBS5_LPBIG_OFFBIG</i>) returns -1 , the meaning of this value is unspecified.
14415		Otherwise, this value is the set of libraries to be given to the <i>cc</i> and <i>c99</i> utilities to build an application using a programming model with an int type using at least 32 bits and long , pointer , and off_t types using at least 64 bits.
14416		
14417		
14418	XSI	_CS_XBS5_LPBIG_OFFBIG_LINTFLAGS (LEGACY)
14419		If <i>sysconf</i> (<i>_SC_XBS5_LPBIG_OFFBIG</i>) returns -1 , the meaning of this value is unspecified.

14420 Otherwise, this value is the set of options to be given to the *lint* utility to check application
 14421 source using a programming model with an **int** type using at least 32 bits and **long**, **pointer**,
 14422 and **off_t** types using at least 64 bits.

14423 The following symbolic constants shall be defined for the *lseek()* and *fcntl()* functions and shall
 14424 have distinct values:

14425 SEEK_CUR Set file offset to current plus *offset*.

14426 SEEK_END Set file offset to EOF plus *offset*.

14427 SEEK_SET Set file offset to *offset*.

14428 The following symbolic constants shall be defined as possible values for the *function* argument
 14429 to the *lockf()* function:

14430 F_LOCK Lock a section for exclusive use.

14431 F_TEST Test section for locks by other processes.

14432 F_TLOCK Test and lock a section for exclusive use.

14433 F_ULOCK Unlock locked sections.

14434 The following symbolic constants shall be defined for *pathconf()*:

14435 ADV _PC_ALLOC_SIZE_MIN

14436 AIO _PC_ASYNC_IO

14437 _PC_CHOWN_RESTRICTED

14438 _PC_FILESIZEBITS

14439 _PC_LINK_MAX

14440 _PC_MAX_CANON

14441 _PC_MAX_INPUT

14442 _PC_NAME_MAX

14443 _PC_NO_TRUNC

14444 _PC_PATH_MAX

14445 _PC_PIPE_BUF

14446 _PC_PRIO_IO

14447 ADV _PC_REC_INCR_XFER_SIZE

14448 _PC_REC_MAX_XFER_SIZE

14449 _PC_REC_MIN_XFER_SIZE

14450 _PC_REC_XFER_ALIGN

14451 _PC_SYNC_IO

14452 _PC_VDISABLE

14453 The following symbolic constants shall be defined for *sysconf()*:

14454 _SC_2_C_BIND

14455 _SC_2_C_DEV

14456 _SC_2_C_VERSION

14457 _SC_2_CHAR_TIME

14458 _SC_2_FORT_DEV

14459 _SC_2_FORT_RUN

14460 _SC_2_LOCALEDEF

14461 _SC_2_PBS

14462 _SC_2_PBS_ACCOUNTING

14463 _SC_2_PBS_CHECKPOINT

14464 _SC_2_PBS_LOCATE

14465 _SC_2_PBS_MESSAGE

14466		_SC_2_PBS_TRACK	
14467		_SC_2_SW_DEV	
14468		_SC_2_UPE	
14469		_SC_2_VERSION	
14470		_SC_ADVISORY_INFO	
14471		_SC_ARG_MAX	
14472		_SC_AIO_LISTIO_MAX	
14473		_SC_AIO_MAX	
14474		_SC_AIO_PRIO_DELTA_MAX	
14475		_SC_ASYNCHRONOUS_IO	
14476	XSI	_SC_ATEXIT_MAX	
14477	BAR	_SC_BARRIERS	
14478		_SC_BASE	
14479		_SC_BC_BASE_MAX	
14480		_SC_BC_DIM_MAX	
14481		_SC_BC_SCALE_MAX	
14482		_SC_BC_STRING_MAX	
14483		_SC_C_LANG_SUPPORT	
14484		_SC_C_LANG_SUPPORT_R	
14485		_SC_CHILD_MAX	
14486		_SC_CLK_TCK	
14487	CS	_SC_CLOCK_SELECTION	
14488		_SC_COLL_WEIGHTS_MAX	
14489		_SC_CPUTIME	
14490		_SC_DELAYTIMER_MAX	
14491		_SC_DEVICE_IO	
14492		_SC_DEVICE_SPECIFIC	
14493		_SC_DEVICE_SPECIFIC_R	
14494		_SC_EXPR_NEST_MAX	
14495		_SC_FD_MGMT	
14496		_SC_FIFO	
14497		_SC_FILE_ATTRIBUTES	
14498		_SC_FILE_LOCKING	
14499		_SC_FILE_SYSTEM	
14500		_SC_FSYNC	
14501		_SC_GETGR_R_SIZE_MAX	
14502		_SC_GETPW_R_SIZE_MAX	
14503		_SC_HOST_NAME_MAX	
14504	XSI	_SC_IOV_MAX	
14505		_SC_JOB_CONTROL	
14506		_SC_LINE_MAX	
14507		_SC_LOGIN_NAME_MAX	
14508		_SC_MAPPED_FILES	
14509		_SC_MEMLOCK	
14510		_SC_MEMLOCK_RANGE	
14511		_SC_MEMORY_PROTECTION	
14512		_SC_MESSAGE_PASSING	
14513	MON	_SC_MONOTONIC_CLOCK	
14514		_SC_MQ_OPEN_MAX	
14515		_SC_MQ_PRIO_MAX	
14516		_SC_MULTIPLE_PROCESS	
14517		_SC_NETWORKING	

14518		_SC_NGROUPS_MAX	
14519		_SC_OPEN_MAX	
14520	XSI	_SC_PAGE_SIZE	
14521		_SC_PAGESIZE	
14522		_SC_PIPE	
14523		_SC_PRIORITIZED_IO	
14524		_SC_PRIORITY_SCHEDULING	
14525		_SC_RE_DUP_MAX	
14526	THR	_SC_READER_WRITER_LOCKS	
14527		_SC_REALTIME_SIGNALS	
14528		_SC_REGEX	
14529		_SC_RTSIG_MAX	
14530		_SC_SAVED_IDS	
14531		_SC_SEMAPHORES	
14532		_SC_SEM_NSEMS_MAX	
14533		_SC_SEM_VALUE_MAX	
14534		_SC_SHARED_MEMORY_OBJECTS	
14535		_SC_SHELL	
14536		_SC_SIGNALS	
14537		_SC_SIGQUEUE_MAX	
14538		_SC_SINGLE_PROCESS	
14539		_SC_SPAWN	
14540	SPI	_SC_SPIN_LOCKS	
14541		_SC_SPORADIC_SERVER	
14542		_SC_STREAM_MAX	
14543		_SC_SYNCHRONIZED_IO	
14544		_SC_SYSTEM_DATABASE	
14545		_SC_SYSTEM_DATABASE_R	
14546		_SC_THREAD_ATTR_STACKADDR	
14547		_SC_THREAD_ATTR_STACKSIZE	
14548		_SC_THREAD_CPUTIME	
14549		_SC_THREAD_DESTRUCTOR_ITERATIONS	
14550		_SC_THREAD_KEYS_MAX	
14551		_SC_THREAD_PRIO_INHERIT	
14552		_SC_THREAD_PRIO_PROTECT	
14553		_SC_THREAD_PRIORITY_SCHEDULING	
14554		_SC_THREAD_PROCESS_SHARED	
14555		_SC_THREAD_SAFE_FUNCTIONS	
14556		_SC_THREAD_SPARADIC_SERVER	
14557		_SC_THREAD_STACK_MIN	
14558		_SC_THREAD_THREADS_MAX	
14559		_SC_TIMEOUTS	
14560		_SC_THREADS	
14561		_SC_TIMER_MAX	
14562		_SC_TIMERS	
14563	TRC	_SC_TRACE	
14564	TEF	_SC_TRACE_EVENT_FILTER	
14565	TRI	_SC_TRACE_INHERIT	
14566	TRL	_SC_TRACE_LOG	
14567		_SC_TTY_NAME_MAX	
14568	TYM	_SC_TYPED_MEMORY_OBJECTS	
14569		_SC_TZNAME_MAX	

```

14570     _SC_USER_GROUPS
14571     _SC_USER_GROUPS_R
14572     _SC_V6_ILP32_OFF32
14573     _SC_V6_ILP32_OFFBIG
14574     _SC_V6_LP64_OFF64
14575     _SC_V6_LPBIG_OFFBIG
14576     _SC_VERSION
14577 XSI   _SC_XBS5_ILP32_OFF32 (LEGACY)
14578     _SC_XBS5_ILP32_OFFBIG (LEGACY)
14579     _SC_XBS5_LP64_OFF64 (LEGACY)
14580     _SC_XBS5_LPBIG_OFFBIG (LEGACY)
14581     _SC_XOPEN_CRYPT
14582     _SC_XOPEN_ENH_I18N
14583     _SC_XOPEN_LEGACY
14584     _SC_XOPEN_REALTIME
14585     _SC_XOPEN_REALTIME_THREADS
14586     _SC_XOPEN_SHM
14587     _SC_XOPEN_STREAMS
14588     _SC_XOPEN_UNIX
14589     _SC_XOPEN_VERSION
14590     _SC_XOPEN_XCU_VERSION
14591

```

14592 The two constants `_SC_PAGESIZE` and `_SC_PAGE_SIZE` may be defined to have the same
14593 value.

14594 The following symbolic constants shall be defined for file streams:

```

14595     STDERR_FILENO    File number of stderr; 2.
14596     STDIN_FILENO     File number of stdin; 0.
14597     STDOUT_FILENO    File number of stdout; 1.

```

14598 Type Definitions

14599 The `size_t`, `ssize_t`, `uid_t`, `gid_t`, `off_t`, and `pid_t` types shall be defined as described in
14600 <sys/types.h>.

14601 The `useconds_t` type shall be defined as described in <sys/types.h>.

14602 The `intptr_t` type shall be defined as described in <inttypes.h>.

14603 Declarations

14604 The following shall be declared as functions and may also be defined as macros. Function
14605 prototypes shall be provided.

```

14606     int             access(const char *, int);
14607     unsigned        alarm(unsigned);
14608     int             chdir(const char *);
14609     int             chown(const char *, uid_t, gid_t);
14610     int             close(int);
14611     size_t          confstr(int, char *, size_t);
14612 XSI   char          *crypt(const char *, const char *);
14613     char          *ctermid(char *);
14614     int             dup(int);

```

```

14615     int          dup2(int, int);
14616 XSI   void          encrypt(char[64], int);
14617     int          execl(const char *, const char *, ...);
14618     int          execlp(const char *, const char *, ...);
14619     int          execlp(const char *, const char *, ...);
14620     int          execv(const char *, char *const []);
14621     int          execve(const char *, char *const [], char *const []);
14622     int          execvp(const char *, char *const []);
14623     void          _exit(int);
14624     int          fchown(int, uid_t, gid_t);
14625 XSI   int          fchdir(int);
14626 SIO   int          fdatsync(int);
14627     pid_t         fork(void);
14628     long          fpathconf(int, int);
14629     int          fsync(int);
14630     int          ftruncate(int, off_t);
14631     char          *getcwd(char *, size_t);
14632     gid_t         getegid(void);
14633     uid_t         geteuid(void);
14634     gid_t         getgid(void);
14635     int          getgroups(int, gid_t []);
14636 XSI   long          gethostid(void);
14637     int          gethostname(char *, size_t);
14638     char          *getlogin(void);
14639     int          getlogin_r(char *, size_t);
14640     int          getopt(int, char * const [], const char *);
14641 XSI   pid_t         getpgid(pid_t);
14642     pid_t         getpgrp(void);
14643     pid_t         getpid(void);
14644     pid_t         getppid(void);
14645 XSI   pid_t         getsid(pid_t);
14646     uid_t         getuid(void);
14647 XSI   char          *getwd(char *); (LEGACY)
14648     int          isatty(int);
14649 XSI   int          lchown(const char *, uid_t, gid_t);
14650     int          link(const char *, const char *);
14651 XSI   int          lockf(int, int, off_t);
14652     off_t         lseek(int, off_t, int);
14653 XSI   int          nice(int);
14654     long          pathconf(const char *, int);
14655     int          pause(void);
14656     int          pipe(int [2]);
14657 XSI   ssize_t        pread(int, void *, size_t, off_t);
14658     ssize_t        pwrite(int, const void *, size_t, off_t);
14659     ssize_t        read(int, void *, size_t);
14660     ssize_t        readlink(const char *restrict, char *restrict, size_t);
14661     int          rmdir(const char *);
14662     int          setegid(gid_t);
14663     int          seteuid(uid_t);
14664     int          setgid(gid_t);

```

```

14665     int          setpgid(pid_t, pid_t);
14666 XSI   pid_t          setpgrp(void);
14667     int          setregid(gid_t, gid_t);
14668     int          setreuid(uid_t, uid_t);
14669     pid_t        setsid(void);
14670     int          setuid(uid_t);
14671     unsigned    sleep(unsigned);
14672 XSI   void          swab(const void *restrict, void *restrict, ssize_t);
14673     int          symlink(const char *, const char *);
14674     void         sync(void);
14675     long         sysconf(int);
14676     pid_t        tcgetpgrp(int);
14677     int          tcsetpgrp(int, pid_t);
14678 XSI   int          truncate(const char *, off_t);
14679     char         *ttyname(int);
14680     int          ttyname_r(int, char *, size_t);
14681 XSI   useconds_t  ualarm(useconds_t, useconds_t);
14682     int          unlink(const char *);
14683 XSI   int          usleep(useconds_t);
14684     pid_t        vfork(void);
14685     ssize_t      write(int, const void *, size_t);

```

14686 Implementations may also include the *pthread_atfork()* prototype as defined in <pthread.h> (on
14687 page 286).

14688 The following external variables shall be declared:

```

14689     extern char   *optarg;
14690     extern int    optind, opterr, optopt;

```

14691 APPLICATION USAGE

14692 IEEE Std 1003.1-200x only describes the behavior of systems that claim conformance to it. |
14693 However, application developers who want to write applications that adapt to other versions of |
14694 IEEE Std 1003.1 (or to systems that do not conform to any POSIX standard) may find it useful to |
14695 code them so as to conditionally compile different code depending on the value of |
14696 `_POSIX_VERSION`, for example. |

```

14697     #if _POSIX_VERSION == 200xxxL |
14698     /* Use the newer function that copes with large files. */ |
14699     off_t pos=ftello(fp); |
14700     #else |
14701     /* Either this is an old version of POSIX, or _POSIX_VERSION is |
14702        not even defined, so use the traditional function. */ |
14703     long pos=ftell(fp); |
14704     #endif |

```

14705 Earlier versions of IEEE Std 1003.1 and of the Single UNIX Specification can be identified by the |
14706 following macros: |

```

14707     POSIX.1-1988 standard |
14708         _POSIX_VERSION==198808L |
14709     POSIX.1-1990 standard |
14710         _POSIX_VERSION==199009L |
14711     ISO POSIX-1:1996 standard |
14712         _POSIX_VERSION==199506L |

```

14713 Single UNIX Specification, Version 1
14714 `_XOPEN_UNIX` and `_XOPEN_VERSION=` =4

14715 Single UNIX Specification, Version 2
14716 `_XOPEN_UNIX` and `_XOPEN_VERSION=` =500

14717 IEEE Std 1003.1-200x does not make any attempt to define application binary interaction with
14718 the underlying operating system. However, application developers may find it useful to query
14719 `_SC_VERSION` at runtime via `sysconf()` to determine whether the current version of the
14720 operating system supports the necessary functionality as in the following program fragment:

```
14721 if (sysconf(_SC_VERSION) < 200xxxL) {  
14722     fprintf(stderr, "POSIX.1-200x system required, terminating \n");  
14723     exit(1);  
14724 }
```

14725 **RATIONALE**

14726 As IEEE Std 1003.1-200x evolved, certain options became sufficiently standardized that it was
14727 concluded that simply requiring one of the option choices was simpler than retaining the option.
14728 However, for backwards compatibility, the option flags (with required constant values) are
14729 retained.

14730 **Version Test Macros**

14731 The standard developers considered altering the definition of `_POSIX_VERSION` and removing
14732 `_SC_VERSION` from the specification of `sysconf()` since the utility to an application was deemed
14733 by some to be minimal, and since the implementation of the functionality is potentially
14734 problematic. However, they recognized that support for existing application binaries is a
14735 concern to manufacturers, application developers, and the users of implementations conforming
14736 to IEEE Std 1003.1-200x.

14737 While the example using `_SC_VERSION` in the APPLICATION USAGE section does not provide
14738 the greatest degree of imaginable utility to the application developer or user, it is arguably better
14739 than a core dump or some other equally obscure result. (It is also possible for implementations
14740 to encode and recognize application binaries compiled in various POSIX.1-conforming
14741 environments, and modify the semantics of the underlying system to conform to the
14742 expectations of the application.) For the reasons outlined in the preceding paragraphs and in the
14743 APPLICATION USAGE section, the standard developers elected to retain the `_POSIX_VERSION`
14744 and `_SC_VERSION` functionality.

14745 **Compile-Time Symbolic Constants for System-Wide Options**

14746 IEEE Std 1003.1-200x now includes support in certain areas for the newly adopted policy
14747 governing options and stubs.

14748 This policy provides flexibility for implementations in how they support options. It also
14749 specifies how conforming applications can adapt to different implementations that support
14750 different sets of options. It allows the following:

- 14751 1. If an implementation has no interest in supporting an option, it does not have to provide
14752 anything associated with that option beyond the announcement that it does not support it.
- 14753 2. An implementation can support a partial or incompatible version of an option (as a non-
14754 standard extension) as long as it does not claim to support the option.
- 14755 3. An application can determine whether the option is supported. A strictly conforming
14756 application must check this announcement mechanism before first using anything
14757 associated with the option.

14758 There is an important implication of this policy. IEEE Std 1003.1-200x cannot dictate the
 14759 behavior of interfaces associated with an option when the implementation does not claim to
 14760 support the option. In particular, it cannot require that a function associated with an
 14761 unsupported option will fail if it does not perform as specified. However, this policy does not
 14762 prevent a standard from requiring certain functions to always be present, but that they shall
 14763 always fail on some implementations. The *setpgid()* function in the POSIX.1-1990 standard, for
 14764 example, is considered appropriate.

14765 The POSIX standards include various options, and the C language binding support for an option
 14766 implies that the implementation must supply data types and function interfaces. An application
 14767 must be able to discover whether the implementation supports each option.

14768 Any application must consider the following three cases for each option:

14769 1. Option never supported.

14770 The implementation advertises at compile time that the option will never be supported. In
 14771 this case, it is not necessary for the implementation to supply any of the data types or
 14772 function interfaces that are provided only as part of the option. The implementation might
 14773 provide data types and functions that are similar to those defined by IEEE Std 1003.1-200x,
 14774 but there is no guarantee for any particular behavior.

14775 2. Option always supported.

14776 The implementation advertises at compile time that the option will always be supported.
 14777 In this case, all data types and function interfaces shall be available and shall operate as
 14778 specified.

14779 3. Option might or might not be supported.

14780 Some implementations might not provide a mechanism to specify support of options at
 14781 compile time. In addition, the implementation might be unable or unwilling to specify
 14782 support or non-support at compile time. In either case, any application that might use the
 14783 option at runtime must be able to compile and execute. The implementation must provide,
 14784 at compile time, all data types and function interfaces that are necessary to allow this. In
 14785 this situation, there must be a mechanism that allows the application to query, at runtime,
 14786 whether the option is supported. If the application attempts to use the option when it is
 14787 not supported, the result is unspecified unless explicitly specified otherwise in
 14788 IEEE Std 1003.1-200x.

14789 FUTURE DIRECTIONS

14790 None.

14791 SEE ALSO

14792 <inttypes.h>, <limits.h>, <sys/socket.h>, <sys/types.h>, <termios.h>, <wctype.h>, the System
 14793 Interfaces volume of IEEE Std 1003.1-200x, *access()*, *alarm()*, *chdir()*, *chown()*, *close()*, *crypt()*,
 14794 *ctermid()*, *dup()*, *encrypt()*, *environ*, *exec()*, *exit()*, *fchdir()*, *fchown()*, *fcntl()*, *fork()*, *fpathconf()*,
 14795 *fsync()*, *ftruncate()*, *getcwd()*, *getegid()*, *geteuid()*, *getgid()*, *getgroups()*, *gethostid()*, *gethostname()*,
 14796 *getlogin()*, *getpgid()*, *getpgrp()*, *getpid()*, *getppid()*, *getsid()*, *getuid()*, *isatty()*, *lchown()*, *link()*,
 14797 *lockf()*, *lseek()*, *nice()*, *pathconf()*, *pause()*, *pipe()*, *read()*, *readlink()*, *rmdir()*, *setgid()*, *setpgid()*,
 14798 *setpgrp()*, *setregid()*, *setreuid()*, *setsid()*, *setuid()*, *sleep()*, *swab()*, *symlink()*, *sync()*, *sysconf()*,
 14799 *tcgetpgrp()*, *tcsetpgrp()*, *truncate()*, *ttyname()*, *ualarm()*, *unlink()*, *usleep()*, *vfork()*, *write()*

14800 CHANGE HISTORY

14801 First released in Issue 1. Derived from Issue 1 of the SVID.

14802 **Issue 5**

14803 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
14804 Threads Extension.

14805 The symbolic constants `_XOPEN_REALTIME` and `_XOPEN_REALTIME_THREADS` are added.
14806 `_POSIX2_C_BIND`, `_XOPEN_ENH_I18N`, and `_XOPEN_SHM` must now be set to a value other
14807 than `-1` by a conforming implementation.

14808 Large File System extensions are added.

14809 The type of the argument to `sbrk()` is changed from `int` to `intptr_t`.

14810 `_XBS_` constants are added to the list of constants for Options and Option Groups, to the list of
14811 constants for the `confstr()` function, and to the list of constants to the `sysconf()` function. These
14812 are all marked EX.

14813 **Issue 6**

14814 `_POSIX2_C_VERSION` is removed.

14815 The Open Group Corrigendum U026/4 is applied, adding the prototype for `fdatasync()`.

14816 The Open Group Corrigendum U026/1 is applied, adding the symbols `_SC_XOPEN_LEGACY`,
14817 `_SC_XOPEN_REALTIME`, and `_SC_XOPEN_REALTIME_THREADS`.

14818 The symbols `_XOPEN_STREAMS` and `_SC_XOPEN_STREAMS` are added to support the XSI
14819 STREAMS Option Group.

14820 Text in the DESCRIPTION relating to conformance requirements is moved elsewhere in |
14821 IEEE Std 1003.1-200x.

14822 The legacy symbol `_SC_PASS_MAX` is removed.

14823 The following new requirements on POSIX implementations derive from alignment with the
14824 Single UNIX Specification:

14825 • The `_CS_POSIX_*` and `_CS_XBS5_*` constants are added for the `confstr()` function. |

14826 • The `_SC_XBS5_*` constants are added for the `sysconf()` function.

14827 • The symbolic constants `F_ULOCK`, `F_LOCK`, `F_TLOCK`, and `F_TEST` are added.

14828 • The `uid_t`, `gid_t`, `off_t`, `pid_t`, and `useconds_t` types are mandated.

14829 The `gethostname()` prototype is added for sockets.

14830 New section added for System Wide Options.

14831 Function prototypes for `setegid()` and `seteuid()` are added.

14832 Option symbolic constants are added for `_POSIX_ADVISORY_INFO`, `_POSIX_CPUTIME`, |
14833 `_POSIX_SPAWN`, `_POSIX_SPORADIC_SERVER`, `_POSIX_THREAD_CPUTIME`,
14834 `_POSIX_THREAD_SPORADIC_SERVER`, and `_POSIX_TIMEOUTS`, and `pathconf()` variables are
14835 added for `_PC_ALLOC_SIZE_MIN`, `_PC_REC_INCR_XFER_SIZE`, `_PC_REC_MAX_XFER_SIZE`,
14836 `_PC_REC_MIN_XFER_SIZE`, and `_PC_REC_XFER_ALIGN` for alignment with
14837 IEEE Std 1003.1d-1999. |

14838 The following are added for alignment with IEEE Std 1003.1j-2000:

14839 • Option symbolic constants `_POSIX_BARRIERS`, `_POSIX_CLOCK_SELECTION`,

14840 `_POSIX_MONOTONIC_CLOCK`, `_POSIX_READER_WRITER_LOCKS`,

14841 `_POSIX_SPIN_LOCKS`, and `_POSIX_TYPED_MEMORY_OBJECTS`

14842 • *sysconf()* variables `_SC_BARRIERS`, `_SC_CLOCK_SELECTION`,
14843 `_SC_MONOTONIC_CLOCK`, `_SC_READER_WRITER_LOCKS`, `_SC_SPIN_LOCKS`, and
14844 `_SC_TYPED_MEMORY_OBJECTS`

14845 The `_SC_XBS5` macros associated with the ISO/IEC 9899:1990 standard are marked LEGACY,
14846 and new equivalent `_SC_V6` macros associated with the ISO/IEC 9899:1999 standard are
14847 introduced.

14848 The *getwd()* function is marked LEGACY.

14849 The **restrict** keyword is added to the prototypes for *readlink()* and *swab()*. |

14850 Constants for options are now harmonized, so when supported they take the year of approval of
14851 IEEE Std 1003.1-200x as the value.

14852 The following are added for alignment with IEEE Std 1003.1q-2000:

14853 • Optional symbolic constants `_POSIX_TRACE`, `_POSIX_TRACE_EVENT_FILTER`,
14854 `_POSIX_TRACE_LOG`, and `_POSIX_TRACE_INHERIT`

14855 • The *sysconf()* symbolic constants `_SC_TRACE`, `_SC_TRACE_EVENT_FILTER`,
14856 `_SC_TRACE_LOG`, and `_SC_TRACE_INHERIT`.

14857 The *brk()* and *sbrk()* legacy functions are removed. |

14858 The Open Group Base Resolution bwg2001-006 is applied, which reworks the XSI versioning |
14859 information. |

14860 The Open Group Base Resolution bwg2001-008 is applied, changing the *namelen* parameter for |
14861 *gethostname()* from **socklen_t** to **size_t**. |

14862 **NAME**14863 `utime.h` — access and modification times structure14864 **SYNOPSIS**14865 `#include <utime.h>`14866 **DESCRIPTION**14867 The **<utime.h>** header shall declare the structure **utimbuf**, which shall include the following
14868 members:14869 `time_t` `actime` Access time.
14870 `time_t` `modtime` Modification time.

14871 The times shall be measured in seconds since the Epoch.

14872 The type **time_t** shall be defined as described in **<sys/types.h>**.14873 The following shall be declared as a function and may also be defined as a macro. A function |
14874 prototype shall be provided. |14875 `int utime(const char *, const struct utimbuf *);`14876 **APPLICATION USAGE**

14877 None.

14878 **RATIONALE**

14879 None.

14880 **FUTURE DIRECTIONS**

14881 None.

14882 **SEE ALSO**14883 **<sys/types.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *utime()*14884 **CHANGE HISTORY**

14885 First released in Issue 3.

14886 **Issue 6**14887 The following new requirements on POSIX implementations derive from alignment with the
14888 Single UNIX Specification:

- 14889
- The **time_t** type is defined.

14890 **NAME**

14891 utmpx.h — user accounting database definitions

14892 **SYNOPSIS**

14893 XSI `#include <utmpx.h>`

14894

14895 **DESCRIPTION**

14896 The <utmpx.h> header shall define the **utmpx** structure that shall include at least the following
 14897 members:

14898	char	ut_user[]	User login name.
14899	char	ut_id[]	Unspecified initialization process identifier.
14900	char	ut_line[]	Device name.
14901	pid_t	ut_pid	Process ID.
14902	short	ut_type	Type of entry.
14903	struct timeval	ut_tv	Time entry was made.

14904 The **pid_t** type shall be defined through **typedef** as described in <sys/types.h>.

14905 The **timeval** structure shall be defined as described in <sys/time.h>.

14906 Inclusion of the <utmpx.h> header may also make visible all symbols from <sys/time.h>.

14907 The following symbolic constants shall be defined as possible values for the *ut_type* member of
 14908 the **utmpx** structure:

14909	EMPTY	No valid user accounting information.
14910	BOOT_TIME	Identifies time of system boot.
14911	OLD_TIME	Identifies time when system clock changed.
14912	NEW_TIME	Identifies time after system clock changed.
14913	USER_PROCESS	Identifies a process.
14914	INIT_PROCESS	Identifies a process spawned by the init process.
14915	LOGIN_PROCESS	Identifies the session leader of a logged in user.
14916	DEAD_PROCESS	Identifies a session leader who has exited.

14917 The following shall be declared as functions and may also be defined as macros. Function |
 14918 prototypes shall be provided. |

```

14919 void          endutxent(void);
14920 struct utmpx *getutxent(void);
14921 struct utmpx *getutxid(const struct utmpx *);
14922 struct utmpx *getutxline(const struct utmpx *);
14923 struct utmpx *pututxline(const struct utmpx *);
14924 void          setutxent(void);
    
```

14925 **APPLICATION USAGE**

14926 None.

14927 **RATIONALE**

14928 None.

14929 **FUTURE DIRECTIONS**

14930 None.

14931 **SEE ALSO**

14932 <sys/time.h>, <sys/types.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *endutxent()*

14933 **CHANGE HISTORY**

14934 First released in Issue 4, Version 2.

14935 **NAME**

14936 `wchar.h` — wide-character handling

14937 **SYNOPSIS**

14938 `#include <wchar.h>`

14939 **DESCRIPTION**

14940 CX Some of the functionality described on this reference page extends the ISO C standard.
 14941 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 14942 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these
 14943 symbols in this header.

14944 The <wchar.h> header shall define the following types:

14945 **wchar_t** As described in <stddef.h>.

14946 **wint_t** An integer type capable of storing any valid value of **wchar_t** or WEOF.

14947 XSI **wctype_t** A scalar type of a data object that can hold values which represent locale-
 14948 specific character classification.

14949 **mbstate_t** An object type other than an array type that can hold the conversion state
 14950 information necessary to convert between sequences of (possibly multi-byte)
 14951 XSI characters and wide characters. If a codeset is being used such that an
 14952 **mbstate_t** needs to preserve more than 2 levels of reserved state, the results
 14953 are unspecified.

14954 XSI **FILE** As described in <stdio.h>.

14955 **size_t** As described in <stddef.h>.

14956 XSI **va_list** As described in <stdarg.h>.

14957 The implementation shall support one or more programming environments in which the width
 14958 of **wint_t** is no greater than the width of type **long**. The names of these programming
 14959 environments can be obtained using the *confstr()* function or the *getconf* utility.

14960 The following shall be declared as functions and may also be defined as macros. Function
 14961 prototypes shall be provided.

```

14962 wint_t      btowc(int);
14963 wint_t      fgetwc(FILE *);
14964 wchar_t     *fgetws(wchar_t *restrict, int, FILE *restrict);
14965 wint_t      fputwc(wchar_t, FILE *);
14966 int         fputws(const wchar_t *restrict, FILE *restrict);
14967 int         fwide(FILE *, int);
14968 int         fwprintf(FILE *restrict, const wchar_t *restrict, ...);
14969 int         fwscanf(FILE *restrict, const wchar_t *restrict, ...);
14970 wint_t      getwc(FILE *);
14971 wint_t      getwchar(void);
14972 XSI int     iswalnum(wint_t);
14973 int         iswalpha(wint_t);
14974 int         iswcntrl(wint_t);
14975 int         iswctype(wint_t, wctype_t);
14976 int         iswdigit(wint_t);
14977 int         iswgraph(wint_t);
14978 int         iswlower(wint_t);
14979 int         iswprint(wint_t);
14980 int         iswpunct(wint_t);
    
```

```

14981     int             iswspace(wint_t);
14982     int             iswupper(wint_t);
14983     int             iswxdigit(wint_t);
14984     size_t          mbrlen(const char *restrict, size_t, mbstate_t *restrict);
14985     size_t          mbrtowc(wchar_t *restrict, const char *restrict, size_t,
14986                          mbstate_t *restrict);
14987     int             mbsinit(const mbstate_t *);
14988     size_t          mbsrtowcs(wchar_t *restrict, const char **restrict, size_t,
14989                          mbstate_t *restrict);
14990     wint_t          putwc(wchar_t, FILE *);
14991     wint_t          putwchar(wchar_t);
14992     int             swprintf(wchar_t *restrict, size_t,
14993                          const wchar_t *restrict, ...);
14994     int             swscanf(const wchar_t *restrict,
14995                          const wchar_t *restrict, ...);
14996 XSI    wint_t          tolower(wint_t);
14997     wint_t          towupper(wint_t);
14998     wint_t          ungetwc(wint_t, FILE *);
14999     int             vfwprintf(FILE *restrict, const wchar_t *restrict, va_list);
15000     int             vfwsscanf(FILE *restrict, const wchar_t *restrict, va_list);
15001     int             vwprintf(const wchar_t *restrict, va_list);
15002     int             vswprintf(wchar_t *restrict, size_t,
15003                          const wchar_t *restrict, va_list);
15004     int             vswscanf(const wchar_t *restrict, const wchar_t *restrict,
15005                          va_list);
15006     int             vwscanf(const wchar_t *restrict, va_list);
15007     size_t          wcrctomb(char *restrict, wchar_t, mbstate_t *restrict);
15008     wchar_t         *wcscat(wchar_t *restrict, const wchar_t *restrict);
15009     wchar_t         *wcschr(const wchar_t *, wchar_t);
15010     int             wcscmp(const wchar_t *, const wchar_t *);
15011     int             wscoll(const wchar_t *, const wchar_t *);
15012     wchar_t         *wcscpy(wchar_t *restrict, const wchar_t *restrict);
15013     size_t          wcscspn(const wchar_t *, const wchar_t *);
15014     size_t          wcsftime(wchar_t *restrict, size_t,
15015                          const wchar_t *restrict, const struct tm *restrict);
15016     size_t          wcslen(const wchar_t *);
15017     wchar_t         *wcsncat(wchar_t *restrict, const wchar_t *restrict, size_t);
15018     int             wcsncmp(const wchar_t *, const wchar_t *, size_t);
15019     wchar_t         *wcsncpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15020     wchar_t         *wcpbrk(const wchar_t *, const wchar_t *);
15021     wchar_t         *wcsrchr(const wchar_t *, wchar_t);
15022     size_t          wcsrtombs(char *restrict, const wchar_t **restrict,
15023                          size_t, mbstate_t *restrict);
15024     size_t          wcsspncpy(const wchar_t *, const wchar_t *);
15025     wchar_t         *wcsstr(const wchar_t *restrict, const wchar_t *restrict);
15026     double          wcstod(const wchar_t *restrict, wchar_t **restrict);
15027     float           wcstof(const wchar_t *restrict, wchar_t **restrict);
15028     wchar_t         *wcstok(wchar_t *restrict, const wchar_t *restrict,
15029                          wchar_t **restrict);
15030     long            wcstol(const wchar_t *restrict, wchar_t **restrict, int);
15031     long double     wcstold(const wchar_t *restrict, wchar_t **restrict);
15032     long long       wcstoll(const wchar_t *restrict, wchar_t **restrict, int);

```

```

15033 unsigned long wcstoul(const wchar_t *restrict, wchar_t **restrict, int);
15034 unsigned long long
15035 wcstoull(const wchar_t *restrict, wchar_t **restrict, int);
15036 XSI wchar_t *wcswcs(const wchar_t *, const wchar_t *);
15037 int wcswidth(const wchar_t *, size_t);
15038 size_t wcsxfrm(wchar_t *restrict, const wchar_t *restrict, size_t);
15039 int wctob(wint_t);
15040 XSI wctype_t wctype(const char *);
15041 int wcwidth(wchar_t);
15042 wchar_t *wmemchr(const wchar_t *, wchar_t, size_t);
15043 int wmemcmp(const wchar_t *, const wchar_t *, size_t);
15044 wchar_t *wmemcpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15045 wchar_t *wmemmove(wchar_t *, const wchar_t *, size_t);
15046 wchar_t *wmemset(wchar_t *, wchar_t, size_t);
15047 int wprintf(const wchar_t *restrict, ...);
15048 int wscanf(const wchar_t *restrict, ...);

```

15049 The <wchar.h> header shall define the following macros:

```

15050 WCHAR_MAX The maximum value representable by an object of type wchar_t.
15051 WCHAR_MIN The minimum value representable by an object of type wchar_t.
15052 WEOF Constant expression of type wint_t that is returned by several WP functions
15053 to indicate end-of-file.
15054 NULL As described in <stddef.h>.

```

15055 The tag **tm** shall be declared as naming an incomplete structure type, the contents of which are
15056 described in the header <time.h>.

15057 CX Inclusion of the <wchar.h> header may make visible all symbols from the headers <ctype.h>,
15058 <stdio.h>, <stdarg.h>, <stdlib.h>, <string.h>, <stddef.h>, and <time.h>.

15059 APPLICATION USAGE

15060 None.

15061 RATIONALE

15062 In the ISO C standard, the symbols referenced as XSI extensions are in <wctype.h>. Their
15063 presence here is thus an extension. |

15064 FUTURE DIRECTIONS

15065 None.

15066 SEE ALSO

15067 <ctype.h>, <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>, the System
15068 Interfaces volume of IEEE Std 1003.1-200x, *btowc()*, *confstr()*, *fgetwc()*, *fgetws()*, *fputwc()*,
15069 *fputws()*, *fwide()*, *fwprintf()*, *fwscanf()*, *getwc()*, *getwchar()*, *iswalnum()*, *iswalpna()*, *iswcntrl()*,
15070 *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*,
15071 *iswxdigit()*, *iswctype()*, *mbsinit()*, *mbrlen()*, *mbrtowc()*, *mbsrtowcs()*, *putwc()*, *putwchar()*,
15072 *swprintf()*, *swscanf()*, *towlower()*, *towupper()*, *ungetwc()*, *vfwprintf()*, *vfwscanf()*, *vswprintf()*,
15073 *vswscanf()*, *vwscanf()*, *wcrtomb()*, *wcsrtombs()*, *wcscat()*, *wcschr()*, *wcscmp()*, *wscoll()*, *wscpy()*,
15074 *wcscspn()*, *wcsftime()*, *wcslen()*, *wcsncat()*, *wcsncmp()*, *wcsncpy()*, *wcsprbk()*, *wcsrchr()*, *wcsspn()*,
15075 *wcsstr()*, *wcstod()*, *wcstof()*, *wcstok()*, *wcstol()*, *wcstold()*, *wcstoll()*, *wcstoul()*, *wcstoull()*, *wcswcs()*,
15076 *wcswidth()*, *wcsxfrm()*, *wctob()*, *wctype()*, *wcwidth()*, *wmemchr()*, *wmemcmp()*, *wmemcpy()*,
15077 *wmemmove()*, *wmemset()*, *wprintf()*, *wscanf()*, the Shell and Utilities volume of
15078 IEEE Std 1003.1-200x, *getconf* |

15079 **CHANGE HISTORY**

15080 First released in Issue 4.

15081 **Issue 5**

15082 Aligned with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

15083 **Issue 6**

15084 The Open Group Corrigendum U021/10 is applied. The prototypes for *wcswidth()* and
15085 *wcwidth()* are marked as extensions.

15086 The Open Group Corrigendum U028/5 is applied, correcting the prototype for the *mbsinit()*
15087 function.

15088 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 15089 • Various function prototypes are updated to add the **restrict** keyword.
- 15090 • The functions *vwscanf()*, *vswscanf()*, *wcstof()*, *wcstold()*, *wcstoll()*, and *wcstoull()* are added.

15091 The type **wctype_t**, the *isw*()*, *to*()*, and *wctype()* functions are marked as XSI extensions.

15092 **NAME**

15093 wctype.h — wide-character classification and mapping utilities

15094 **SYNOPSIS**

15095 #include <wctype.h>

15096 **DESCRIPTION**

15097 **CX** Some of the functionality described on this reference page extends the ISO C standard.
 15098 Applications shall define the appropriate feature test macro (see the System Interfaces volume of
 15099 IEEE Std 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of these
 15100 symbols in this header.

15101 The <wctype.h> header shall define the following types:

15102 **wint_t** As described in <wchar.h>.

15103 **wctrans_t** A scalar type that can hold values which represent locale-specific character
 15104 mappings.

15105 **wctype_t** As described in <wchar.h>.

15106 The following shall be declared as functions and may also be defined as macros. Function
 15107 prototypes shall be provided.

```

15108 int      iswalnum(wint_t);
15109 int      iswalpha(wint_t);
15110 int      iswblank(wint_t);
15111 int      iswcntrl(wint_t);
15112 int      iswdigit(wint_t);
15113 int      iswgraph(wint_t);
15114 int      iswlower(wint_t);
15115 int      iswprint(wint_t);
15116 int      iswpunct(wint_t);
15117 int      iswspace(wint_t);
15118 int      iswupper(wint_t);
15119 int      iswxdigit(wint_t);
15120 int      iswctype(wint_t, wctype_t);
15121 wint_t   towctrans(wint_t, wctrans_t);
15122 wint_t   tolower(wint_t);
15123 wint_t   toupper(wint_t);
15124 wctrans_t wctrans(const char *);
15125 wctype_t wctype(const char *);
    
```

15126 The <wctype.h> header shall define the following macro name:

15127 **WEOF** Constant expression of type **wint_t** that is returned by several MSE functions
 15128 to indicate end-of-file.

15129 For all functions described in this header that accept an argument of type **wint_t**, the value is
 15130 representable as a **wchar_t** or equals the value of **WEOF**. If this argument has any other value,
 15131 the behavior is undefined.

15132 The behavior of these functions shall be affected by the *LC_CTYPE* category of the current locale.

15133 **CX** Inclusion of the <wctype.h> header may make visible all symbols from the headers <ctype.h>,
 15134 <stdio.h>, <stdarg.h>, <stdlib.h>, <string.h>, <stddef.h>, <time.h>, and <wchar.h>.

15135 **APPLICATION USAGE**

15136 None.

15137 **RATIONALE**

15138 None.

15139 **FUTURE DIRECTIONS**

15140 None.

15141 **SEE ALSO**

15142 <locale.h>, <wchar.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *iswalnum()*,
15143 *iswalph()*, *iswblank()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,
15144 *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *towctrans()*, *towlower()*, *towupper()*,
15145 *wctrans()*, *wctype()*

15146 **CHANGE HISTORY**

15147 First released in Issue 5. Derived from the ISO/IEC 9899:1990/Amendment 1:1995 (E).

15148 **Issue 6**

15149 The *iswblank()* function is added for alignment with the ISO/IEC 9899:1999 standard.

15150 NAME

15151 wordexp.h — word-expansion types

15152 SYNOPSIS

15153 #include <wordexp.h>

15154 DESCRIPTION

15155 The <wordexp.h> header shall define the structures and symbolic constants used by the
15156 *wordexp()* and *wordfree()* functions.

15157 The structure type **wordexp_t** shall contain at least the following members:

15158 size_t we_wordc Count of words matched by *words*.
15159 char **we_wordv Pointer to list of expanded words.
15160 size_t we_offs Slots to reserve at the beginning of *we_wordv*.

15161 The *flags* argument to the *wordexp()* function shall be the bitwise-inclusive OR of the following
15162 flags:

15163 WRDE_APPEND Append words to those previously generated.
15164 WRDE_DOOFFS Number of null pointers to prepend to *we_wordv*.
15165 WRDE_NOCMD Fail if command substitution is requested.
15166 WRDE_REUSE The *pwordexp* argument was passed to a previous successful call to
15167 *wordexp()*, and has not been passed to *wordfree()*. The result is the same
15168 as if the application had called *wordfree()* and then called *wordexp()*
15169 without WRDE_REUSE.
15170 WRDE_SHOWERR Do not redirect *stderr* to */dev/null*.
15171 WRDE_UNDEF Report error on an attempt to expand an undefined shell variable.

15172 The following constants shall be defined as error return values:

15173 WRDE_BADCHAR One of the unquoted characters—<newline>, ' | ', '&', ';', '<', '>',
15174 '(', ') ', '{ ', '}'—appears in *words* in an inappropriate context.
15175 WRDE_BADVAL Reference to undefined shell variable when WRDE_UNDEF is set in *flags*.
15176 WRDE_CMDSUB Command substitution requested when WRDE_NOCMD was set in *flags*.
15177 WRDE_NOSPACE Attempt to allocate memory failed.
15178 OB XSI WRDE_NOSYS Reserved.
15179 WRDE_SYNTAX Shell syntax error, such as unbalanced parentheses or unterminated
15180 string.

15181 The <wordexp.h> header shall define the following type: |

15182 XSI **size_t** As described in <stddef.h>. |

15183 The following shall be declared as functions and may also be defined as macros. Function |
15184 prototypes shall be provided. |

15185 int wordexp(const char *restrict, wordexp_t *restrict, int);
15186 void wordfree(wordexp_t *);

15187 The implementation may define additional macros or constants using names beginning with
15188 WRDE_.

15189 **APPLICATION USAGE**

15190 None.

15191 **RATIONALE**

15192 None.

15193 **FUTURE DIRECTIONS**

15194 None.

15195 **SEE ALSO**

15196 <stddef.h>, the System Interfaces volume of IEEE Std 1003.1-200x, *wordexp()*, the Shell and |

15197 Utilities volume of IEEE Std 1003.1-200x

15198 **CHANGE HISTORY**

15199 First released in Issue 4. Derived from the ISO POSIX-2 standard.

15200 **Issue 6**

15201 The **restrict** keyword is added to the prototype for *wordexp()*.

15202 The WRDE_NOSYS constant is marked obsolescent.