

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

- 19 • Graphics interfaces
- 20 • Database management system interfaces
- 21 • Record I/O considerations
- 22 • Object or binary code portability
- 23 • System configuration and resource availability

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

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

- 29 • IEEE Std. 1003.1-1996 (POSIX-1) (incorporating IEEE Stds. 1003.1-1990, 1003.1b-1993,
30 1003.1c-1995, and 1003.1i-1995)
- 31 • The following amendments to the POSIX.1-1990 standard:
 - 32 — IEEE P1003.1a draft standard (Additional System Services)
 - 33 — IEEE Std. 1003.1d-1999 (Additional Realtime Extensions)
 - 34 — IEEE Std. 1003.1g-2000 (Protocol-Independent Interfaces (PII))
 - 35 — IEEE Std. 1003.1j-2000 (Advanced Realtime Extensions)
 - 36 — IEEE Std. 1003.1q-2000 (Tracing)

- 37 • IEEE Std. 1003.2-1992 (POSIX-2) (includes IEEE Std. 1003.2a-1992)
- 38 • The following amendment to the ISO POSIX-2: 1993 standard:
- 39 — IEEE P1003.2b draft standard (Additional Utilities)
- 40 — IEEE Std. 1003.2d-1994 (Batch Environment)
- 41 • Open Group Technical Standard, February 1997, System Interface Definitions, Issue 5 (XBD5)
- 42 (ISBN: 1-85912-186-1, C605)
- 43 • Open Group Technical Standard, February 1997, Commands and Utilities, Issue 5 (XCU5)
- 44 (ISBN: 1-85912-191-8, C604)
- 45 • Open Group Technical Standard, February 1997, System Interfaces and Headers, Issue 5
- 46 (XSH5) (in 2 Volumes) (ISBN: 1-85912-181-0, C606)
- 47 **Note:** XBD5, XCU5, and XSH5 are collectively referred to as the *Base Specifications*.
- 48 • Open Group Technical Standard, January 2000, Networking Services, Issue 5.2 (XNS5.2)
- 49 (ISBN: 1-85912-241-8, C808)
- 50 • ISO/IEC 9899: 1999, Programming Languages — C.
- 51 IEEE Std. 1003.1-200x uses the *Base Specifications* as its organizational basis and adds the
- 52 following additional functionality to them drawn from the base documents above:
- 53 • Normative text from the ISO POSIX-1: 1996 standard and the ISO POSIX-2: 1993 standard not
- 54 included in the *Base Specifications*
- 55 • The amendments to the POSIX.1-1990 standard and the ISO POSIX-2: 1993 standard listed
- 56 above, except for parts of IEEE Std. 1003.1g-2000
- 57 • Portability Considerations
- 58 • Additional rationale and notes
- 59 The following features, marked legacy or obsolescent in the base documents, are not carried
- 60 forward into IEEE Std. 1003.1-200x. Other features from the base documents marked legacy or
- 61 obsolescent are carried forward unless otherwise noted.
- 62 From XSH5, the following legacy interfaces, headers, and external variables are not carried
- 63 forward:
- 64 *advance()*, *brk()*, *chroot()*, *compile()*, *cuserid()*, *gamma()*, *getdtablesize()*, *getpagesize()*, *getpass()*,
- 65 *getw()*, *putw()*, *re_comp()*, *re_exec()*, *regcmp()*, *sbrk()*, *sigstack()*, *wait3()*, *<re_comp.h>*,
- 66 *<regexp.h>*, *<varargs.h>*, *loc1*, *__loc1*, *loc2*, *locs*
- 67 From XCU5, the following legacy utilities are not carried forward:
- 68 *calendar*, *cancel*, *cc*, *col*, *cpio*, *cu*, *dircmp*, *dis*, *egrep*, *fgrep*, *line*, *lint*, *lpstat*, *mail*, *pack*, *pcat*, *pg*, *spell*,
- 69 *sum*, *tar*, *unpack*, *uulog*, *uname*, *uupick*, *uuto*
- 70 In addition, legacy features within non-legacy reference pages (for example, headers) are not
- 71 carried forward.
- 72 From the ISO POSIX-1: 1996 standard, the following obsolescent features are not carried
- 73 forward:
- 74 Page 112, CLK_TCK
- 75 Page 197 *tgetattr()* rate returned option
- 76 From the ISO POSIX-2: 1993 standard, obsolescent features within the following pages are not
- 77 carried forward:

78 Page 75 zero-length prefix within PATH
 79 Page 156, 159 *set*,
 80 Page 178, *awk*, use of no argument and no parentheses with length
 81 Page 259, *ed*
 82 Page 272, *env*
 83 Page 282, *find -perm[-]onum*,
 84 Page 295-296, *egrep*
 85 Page 299-300, *head*
 86 Page 305-306, *join*
 87 Page 309-310, *kill*
 88 Page 431-433, 435-436, *sort*
 89 Page 444-445, *tail*
 90 Page 453, 455-456, *touch*
 91 Page 464-465, *tty*
 92 Page 472, *uniq*
 93 Page 515-516, *ex*
 94 Page 542-543, *expand*
 95 Page 563-565, *more*
 96 Page 574-576, *newgrp*
 97 Page 578, *nice*
 98 Page 594-596, *renice*
 99 Page 597-598, *split*
 100 Page 600-601, *strings*
 101 Page 624-625, *vi*
 102 Page 693, *lex*

103 The *c89* utility (which specified a compiler for the C Language specified by the
 104 ISO/IEC 9899:1990 standard) has been replaced by a *c99* utility (which specifies a compiler for
 105 the C Language specified by the ISO/IEC 9899:1999 standard).

106 From XSH5, text marked OH has been reviewed on a case-by-case basis and removed where
 107 appropriate. The XCU5 text marked OF, OP, PI, and UN has been reviewed on a case-by-case
 108 basis and removed where appropriate

109 For the networking interfaces, the base document is the XNS, Issue 5.2 specification. The
 110 following parts of the XNS, Issue 5.2 specification are out of scope and not included in
 111 IEEE Std. 1003.1-200x:

- 112 • Part 3 (XTI)
- 113 • Part 4 (Appendixes)

114 Since there is much duplication between the XNS, Issue 5.2 specification and
 115 IEEE Std. 1003.1g-2000, material only from the following sections of IEEE Std. 1003.1g-2000 has
 116 been considered for inclusion:

- 117 • General terms related to sockets (Clause 2.2.2)
- 118 • Socket concepts (Clauses 5.1 through 5.3, inclusive)
- 119 • The *pselect()* function (Clauses 6.2.2.1 and 6.2.3)
- 120 • The *isfdtype()* function (Clause 5.4.8)
- 121 • The `<sys/select.h>` header (Clause 6.2)

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

124	• Issues raised by IEEE or ISO/IEC Interpretations against IEEE Std. 1003.1 and IEEE Std. 1003.2	
125		
126	• Issues raised in corrigenda for the <i>Base Specifications</i> and working group resolutions from The Open Group	
127		
128	• Corrigenda and resolutions passed by The Open Group for the XNS, Issue 5.2 specification	
129	• Changes to make the text self-consistent with the additional material merged	
130	• A reorganization of the options in order to facilitate profiling, both for smaller profiles such as IEEE Std 1003.13, and larger profiles such as the Single UNIX Specification	
131		
132	• Alignment with the ISO/IEC 9899:1999 standard	

133 **1.2 Conformance**

134 Conformance requirements for IEEE Std. 1003.1-200x are defined in Chapter 2 (on page 19).

135 1.3 Normative References

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

142 **Notes to Reviewers**

143 *This section with side shading will not appear in the final copy. - Ed.*

144 The following list will be updated.

145 ANS X3.9-1978

146 (Reaffirmed 1989) American National Standard Programming Language FORTRAN.¹

147 ISO/IEC 646

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

150 ISO 4217

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

152 ISO/IEC 4873

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

155 ISO 8601

156 ISO 8601: 1988, Data Elements and Interchange Formats — Information Interchange —
 157 Representation of Dates and Times.

158 ISO 8859-1

159 ISO 8859-1: 1988, Information Processing — 8-bit Single-byte Coded Graphic Character Sets
 160 — Part 1: Latin Alphabet No. 1.

161 ISO 8859-2

162 ISO 8859-2: 1987, Information Processing — 8-bit Single-byte Coded Graphic Character Sets
 163 — Part 2: Latin Alphabet No. 2.

164 ISO/IEC 9899

165 ISO/IEC 9899: 1999, Programming Languages — C.

166 ISO/IEC 9945-1

167 ISO/IEC 9945-1: 200x, Information Technology — Portable Operating System Interface
 168 (POSIX) — Part 1: System Application Program Interface (API) [C Language] (identical to
 169 ANSI/IEEE Std 1003.1-200x).³

170

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

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

175 3. This standard is available from the IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, U.S.A. Tel:
 176 1 (800) 678-IEEE or +1 (908) 981-1393.

177	ISO/IEC 9945-2
178	ISO/IEC 9945-2:1993, Information Technology — Portable Operating System Interface
179	(POSIX) — Part 2: Shell and Utilities.
180	ISO/IEC 10646-1
181	ISO/IEC 10646-1:1993, Information Technology — Universal Multiple-Octet Coded
182	Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.
183	ISO/IEC 14519:1999
184	ISO/IEC 14519:1999, Information Technology — POSIX Ada Language Interfaces —
185	Binding for System Application Program Interface (API) — Realtime Extensions.

186 1.4 Terminology

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

188 **can**

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

192 **implementation-defined**

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

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

200 **legacy**

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

205 **may**

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

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

211 **shall**

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

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

216 **should**

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

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

223 **undefined**

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

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

230	unspecified	
231		Describes the nature of a value or behavior not specified by IEEE Std. 1003.1-200x which
232		results from use of a valid program construct or valid data input.
233		The value or behavior may vary among implementations that conform to
234		IEEE Std. 1003.1-200x. An application should not rely on the existence or validity of the
235		value or behavior. An application that relies on any particular value or behavior cannot be
236		assured to be portable across conforming implementations.

237 1.5 Portability

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

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

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

251 1.5.1 Codes

252 The codes and their meanings are as follows. See also Section 1.5.2 (on page 17).

253 ADV **Advisory Information**

254 The functionality described is optional. The functionality described is also an extension to the
 255 ISO C standard.

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

259 AIO **Asynchronous Input and Output**

260 The functionality described is optional. The functionality described is also an extension to the
 261 ISO C standard.

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

265 BAR **Barriers**

266 The functionality described is optional. The functionality described is also an extension to the
 267 ISO C standard.

268 Where applicable, functions are marked with the BAR margin legend in the SYNOPSIS section.
 269 Where additional semantics apply to a function, the material is identified by use of the BAR
 270 margin legend.

271 BE **Batch Environment Services and Utilities**

272 The functionality described is optional.

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

276 CD **C-Language Development Utilities**

277 The functionality described is optional.

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

281	CPT	Process CPU-Time Clocks
282		The functionality described is optional. The functionality described is also an extension to the
283		ISO C standard.
284		Where applicable, functions are marked with the CPT margin legend in the SYNOPSIS section.
285		Where additional semantics apply to a function, the material is identified by use of the CPT
286		margin legend.
287	CS	Clock Selection
288		The functionality described is optional. The functionality described is also an extension to the
289		ISO C standard.
290		Where applicable, functions are marked with the CS margin legend in the SYNOPSIS section.
291		Where additional semantics apply to a function, the material is identified by use of the CS
292		margin legend.
293	CX	Extension to the ISO C standard
294		The functionality described is an extension to the ISO C standard. Application writers may
295		make use of an extension as it is supported on all IEEE Std. 1003.1-200x-conforming systems.
296	FD	FORTTRAN Development Utilities
297		The functionality described is optional.
298		Where applicable, utilities are marked with the FD margin legend in the SYNOPSIS section.
299		Where additional semantics apply to a utility, the material is identified by use of the FD margin
300		legend.
301	FR	FORTTRAN Runtime Utilities
302		The functionality described is optional.
303		Where applicable, utilities are marked with the FR margin legend in the SYNOPSIS section.
304		Where additional semantics apply to a utility, the material is identified by use of the FR margin
305		legend.
306	FSC	File Synchronization
307		The functionality described is optional. The functionality described is also an extension to the
308		ISO C standard.
309		Where applicable, functions are marked with the FSC margin legend in the SYNOPSIS section.
310		Where additional semantics apply to a function, the material is identified by use of the FSC
311		margin legend.
312	IP6	IPV6
313		The functionality described is optional. The functionality described is also an extension to the
314		ISO C standard.
315		Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section.
316		Where additional semantics apply to a function, the material is identified by use of the IP6
317		margin legend.
318	MAN	Mandatory in the Next Draft
319		This is an interim draft code used to aid reviewers during the development of
320		IEEE Std. 1003.1-200x. It denotes a feature that was previously an option or extension that is
321		being brought into the mandatory base functionality. This margin code will be removed from the
322		final draft.
323	MF	Memory Mapped Files
324		The functionality described is optional. The functionality described is also an extension to the
325		ISO C standard.

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

329 ML **Process Memory Locking**

330 The functionality described is optional. The functionality described is also an extension to the
331 ISO C standard.

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

335 MLR **Range Memory Locking**

336 The functionality described is optional. The functionality described is also an extension to the
337 ISO C standard.

338 Where applicable, functions are marked with the MLR margin legend in the SYNOPSIS section.
339 Where additional semantics apply to a function, the material is identified by use of the MLR
340 margin legend.

341 MON **Monotonic Clock**

342 The functionality described is optional. The functionality described is also an extension to the
343 ISO C standard.

344 Where applicable, functions are marked with the MON margin legend in the SYNOPSIS section.
345 Where additional semantics apply to a function, the material is identified by use of the MON
346 margin legend.

347 MPR **Memory Protection**

348 The functionality described is optional. The functionality described is also an extension to the
349 ISO C standard.

350 Where applicable, functions are marked with the MPR margin legend in the SYNOPSIS section.
351 Where additional semantics apply to a function, the material is identified by use of the MPR
352 margin legend.

353 MSG **Message Passing**

354 The functionality described is optional. The functionality described is also an extension to the
355 ISO C standard.

356 Where applicable, functions are marked with the MSG margin legend in the SYNOPSIS section.
357 Where additional semantics apply to a function, the material is identified by use of the MSG
358 margin legend.

359 OB **Obsolescent**

360 The functionality described may be withdrawn in a future version of this volume of
361 IEEE Std. 1003.1-200x. Strictly Conforming POSIX Applications and Strictly Conforming XSI
362 Applications shall not use obsolescent features.

363 OF **Output Format Incompletely Specified**

364 The functionality described is an XSI extension. The format of the output produced by the utility
365 is not fully specified. It is therefore not possible to post-process this output in a consistent
366 fashion. Typical problems include unknown length of strings and unspecified field delimiters.

367 OH **Optional Header**

368 In the SYNOPSIS section of some interfaces in the System Interfaces volume of
369 IEEE Std. 1003.1-200x an included header is marked as in the following example:

```

370 OH      #include <sys/types.h>
371          #include <grp.h>
372          struct group *getgrnam(const char *name);

```

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

374 PIO **Prioritized Input and Output**
375 The functionality described is optional. The functionality described is also an extension to the
376 ISO C standard.

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

380 PS **Process Scheduling**
381 The functionality described is optional. The functionality described is also an extension to the
382 ISO C standard.

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

386 RTS **Realtime Signals Extension**
387 The functionality described is optional. The functionality described is also an extension to the
388 ISO C standard.

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

392 SD **Software Development Utilities**
393 The functionality described is optional.

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

397 SEM **Semaphores**
398 The functionality described is optional. The functionality described is also an extension to the
399 ISO C standard.

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

403 SHM **Shared Memory Objects**
404 The functionality described is optional. The functionality described is also an extension to the
405 ISO C standard.

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

409 SIO **Synchronized Input and Output**
410 The functionality described is optional. The functionality described is also an extension to the
411 ISO C standard.

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

415	SPI	Spin Locks
416		The functionality described is optional. The functionality described is also an extension to the
417		ISO C standard.
418		Where applicable, functions are marked with the SPI margin legend in the SYNOPSIS section.
419		Where additional semantics apply to a function, the material is identified by use of the SPI
420		margin legend.
421	SPN	Spawn
422		The functionality described is optional. The functionality described is also an extension to the
423		ISO C standard.
424		Where applicable, functions are marked with the SPN margin legend in the SYNOPSIS section.
425		Where additional semantics apply to a function, the material is identified by use of the SPN
426		margin legend.
427	SS	Process Sporadic Server
428		The functionality described is optional. The functionality described is also an extension to the
429		ISO C standard.
430		Where applicable, functions are marked with the SS margin legend in the SYNOPSIS section.
431		Where additional semantics apply to a function, the material is identified by use of the SS
432		margin legend.
433	TCT	Thread CPU-Time Clocks
434		The functionality described is optional. The functionality described is also an extension to the
435		ISO C standard.
436		Where applicable, functions are marked with the TCT margin legend in the SYNOPSIS section.
437		Where additional semantics apply to a function, the material is identified by use of the TCT
438		margin legend.
439	THR	Threads
440		The functionality described is optional. The functionality described is also an extension to the
441		ISO C standard.
442		Where applicable, functions are marked with the THR margin legend in the SYNOPSIS section.
443		Where additional semantics apply to a function, the material is identified by use of the THR
444		margin legend.
445	TMO	Timeouts
446		The functionality described is optional. The functionality described is also an extension to the
447		ISO C standard.
448		Where applicable, functions are marked with the TMO margin legend in the SYNOPSIS section.
449		Where additional semantics apply to a function, the material is identified by use of the TMO
450		margin legend.
451	TMR	Timers
452		The functionality described is optional. The functionality described is also an extension to the
453		ISO C standard.
454		Where applicable, functions are marked with the TMR margin legend in the SYNOPSIS section.
455		Where additional semantics apply to a function, the material is identified by use of the TMR
456		margin legend.
457	TPI	Threads Priority Inheritance
458		The functionality described is optional. The functionality described is also an extension to the
459		ISO C standard.

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

463 TPP **Thread Priority Protection**

464 The functionality described is optional. The functionality described is also an extension to the
465 ISO C standard.

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

469 TPS **Thread Execution Scheduling**

470 The functionality described is optional. The functionality described is also an extension to the
471 ISO C standard.

472 Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
473 Where additional semantics apply to a function, the material is identified by use of the TPS
474 margin legend.

475 TRC **Trace**

476 The functionality described is optional. The functionality described is also an extension to the
477 ISO C standard.

478 Where applicable, functions are marked with the TRC margin legend in the SYNOPSIS section.
479 Where additional semantics apply to a function, the material is identified by use of the TRC
480 margin legend.

481 TEF **Trace Event Filter**

482 The functionality described is optional. The functionality described is also an extension to the
483 ISO C standard.

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

487 TRL **Trace Log**

488 The functionality described is optional. The functionality described is also an extension to the
489 ISO C standard.

490 Where applicable, functions are marked with the TRL margin legend in the SYNOPSIS section.
491 Where additional semantics apply to a function, the material is identified by use of the TRL
492 margin legend.

493 TRI **Trace Inherit**

494 The functionality described is optional. The functionality described is also an extension to the
495 ISO C standard.

496 Where applicable, functions are marked with the TRI margin legend in the SYNOPSIS section.
497 Where additional semantics apply to a function, the material is identified by use of the TRI
498 margin legend.

499 TSA **Thread Stack Address Attribute**

500 The functionality described is optional. The functionality described is also an extension to the
501 ISO C standard.

502 Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
503 Where additional semantics apply to a function, the material is identified by use of the TSA
504 margin legend.

505	TSF	Thread-Safe Functions
506		The functionality described is optional. The functionality described is also an extension to the
507		ISO C standard.
508		Where applicable, functions are marked with the TSF margin legend in the SYNOPSIS section.
509		Where additional semantics apply to a function, the material is identified by use of the TSF
510		margin legend.
511	TSH	Thread Process-Shared Synchronization
512		The functionality described is optional. The functionality described is also an extension to the
513		ISO C standard.
514		Where applicable, functions are marked with the TSH margin legend in the SYNOPSIS section.
515		Where additional semantics apply to a function, the material is identified by use of the TSH
516		margin legend.
517	TSP	Thread Sporadic Server
518		The functionality described is optional. The functionality described is also an extension to the
519		ISO C standard.
520		Where applicable, functions are marked with the TSP margin legend in the SYNOPSIS section.
521		Where additional semantics apply to a function, the material is identified by use of the TSP
522		margin legend.
523	TSS	Thread Stack Address Size
524		The functionality described is optional. The functionality described is also an extension to the
525		ISO C standard.
526		Where applicable, functions are marked with the TSS margin legend in the SYNOPSIS section.
527		Where additional semantics apply to a function, the material is identified by use of the TSS
528		margin legend.
529	TYM	Typed Memory Objects
530		The functionality described is optional. The functionality described is also an extension to the
531		ISO C standard.
532		Where applicable, functions are marked with the TYM margin legend in the SYNOPSIS section.
533		Where additional semantics apply to a function, the material is identified by use of the TYM
534		margin legend.
535	UN	Possibly Unsupportable Feature
536		The functionality described is an XSI extension. It need not be possible to implement the
537		required functionality (as defined) on all conformant systems and the functionality need not be
538		present. This may, for example, be the case where the conformant system is hosted and the
539		underlying system provides the service in an alternative way.
540	UP	User Portability Utilities
541		The functionality described is optional.
542		Where applicable, utilities are marked with the UP margin legend in the SYNOPSIS section.
543		Where additional semantics apply to a utility, the material is identified by use of the UP margin
544		legend.
545	XSI	Extension
546		The functionality described is an XSI extension. Functionality marked XSI is also an extension to
547		the ISO C standard. Application writers may confidently make use of an extension on all
548		systems supporting the X/Open System Interfaces Extension.

549 If an entire SYNOPSIS section is shaded and marked with one XSI, all the functionality described
 550 in that reference page is an extension. See Section 3.441 (on page 117).

551 XSR **XSI STREAMS**
 552 The functionality described is optional. The functionality described is also an extension to the
 553 ISO C standard.

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

557 1.5.2 Margin Code Notation

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

- 561 • A feature dependent on one or two options.

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

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

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

- 567 • A feature dependent on either of the options denoted.

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

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

- 572 • A feature dependent on more than two options.

573 The following special notations are used:

574 code1 ADV (MF | SHM) This feature requires support of the Advisory Informtion option and either
 575 the Memory Mapped Files or Shared Memory Objects option.

576 code2 MF|SHM|MPR This feature requires support of either the Memory Mapped Files, Shared
 577 Memory Objects, or Memory Protection options.

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

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

584 2.1 Implementation Conformance

585 2.1.1 Requirements

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

- 587 1. The system shall support all utilities, functions, and facilities defined within
588 IEEE Std. 1003.1-200x that are required for POSIX conformance (see Section 2.1.3 (on page
589 20)). These interfaces shall support the functional behavior described herein.
- 590 2. The system may support one or more options as described under Section 2.1.5 (on page
591 25). When an implementation claims that an option is supported, all of its constituent
592 parts shall be provided.
- 593 3. The system may support the X/Open System Interface Extension (XSI) as described under
594 Section 2.1.4 (on page 23).
- 595 4. The system may provide additional utilities, functions, or facilities not required by
596 IEEE Std. 1003.1-200x. Non-standard extensions of the utilities, functions, or facilities
597 specified in IEEE Std. 1003.1-200x should be identified as such in the system
598 documentation. Non-standard extensions, when used, may change the behavior of utilities,
599 functions, or facilities defined by IEEE Std. 1003.1-200x. The conformance document shall
600 define an environment in which an application can be run with the behavior specified by
601 IEEE Std. 1003.1-200x. In no case shall such an environment require modification of a
602 Strictly Conforming POSIX Application (see Section 2.2.1 (on page 38)).

603 2.1.2 Documentation

604 A conformance document with the following information shall be available for an
605 implementation claiming conformance to IEEE Std. 1003.1-200x. The conformance document
606 shall have the same structure as IEEE Std. 1003.1-200x, with the information presented in the
607 appropriate sections and subsections. Sections and subsections that consist solely of subordinate
608 section titles, with no other information, are not required. The conformance document shall not
609 contain information about extended facilities or capabilities outside the scope of
610 IEEE Std. 1003.1-200x.

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

616 The conformance document shall describe the limit values found in the headers `<limits.h>` (on
617 page 281) and `<unistd.h>` (on page 437), stating values, the conditions under which those values
618 may change, and the limits of such variations, if any.

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

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

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

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

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

635 The system documentation should also contain the information found in the conformance
636 document.

637 **2.1.3 POSIX Conformance**

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

639 *2.1.3.1 POSIX System Interfaces*

- 640 • The system shall set the symbolic constant `_POSIX_BASE` to a value other than `-1`. |
- 641 • The system shall support the following symbolic constants, reflecting mandatory Profiling
642 Option Groups for IEEE Std. 1003.1-200x (see Section 2.1.5 (on page 25)):
 - 643 — `_POSIX_C_LANG_SUPPORT`
 - 644 — `_POSIX_DEVICE_IO`
 - 645 — `_POSIX_DEVICE_SPECIFIC`
 - 646 — `_POSIX_FD_MGMT`
 - 647 — `_POSIX_FIFO`
 - 648 — `_POSIX_FILE_ATTRIBUTES`
 - 649 — `_POSIX_FILE_SYSTEM`
 - 650 — `_POSIX_JOB_CONTROL`
 - 651 — `_POSIX_MULTIPLE_PROCESS`
 - 652 — `_POSIX_PIPE`
 - 653 — `_POSIX_SIGNALS`
 - 654 — `_POSIX_SINGLE_PROCESS`
 - 655 — `_POSIX_SYSTEM_DATABASE`
 - 656 — `_POSIX_USER_GROUPS`
 - 657 — `_POSIX_NETWORKING`
- 658 • The system may support one or more Profiling Option Groups (see Section 2.1.5.1 (on page
659 25)) denoted by the following symbolic constants:

- 660 — `_POSIX_C_LANG_SUPPORT_R`
 661 — `_POSIX_FILE_LOCKING`
 662 — `_POSIX_SYSTEM_DATABASE_R`
 663 — `_POSIX_USER_GROUPS_R`
- 664 • Although all implementations conforming to IEEE Std. 1003.1-200x support all the features
 665 described below, there may be system-dependent or file system-dependent configuration
 666 procedures that can remove or modify any or all of these features. Such configurations
 667 should not be made if strict compliance is required.
- 668 The following symbolic constants shall either be undefined or defined with a value other
 669 than `-1`. If a constant is undefined, an application should use the `sysconf()`, `pathconf()`, or
 670 `fpathconf()` functions, or the `getconf` utility, to determine which features are present on the
 671 system at that time or for the particular path name in question.
- 672 — `_POSIX_CHOWN_RESTRICTED`
- 673 The use of `chown()` is restricted to a process with appropriate privileges, and to changing
 674 the group ID of a file only to the effective group ID of the process or to one of its
 675 supplementary group IDs.
- 676 — `_POSIX_NO_TRUNC`
- 677 Path name components longer than `{NAME_MAX}` generate an error.
- 678 • The following symbolic constants shall be defined with a value other than `-1`:
 679 — `_POSIX_JOB_CONTROL`
 680 — `_POSIX_SAVED_IDS`
 681 — `_POSIX_VDISABLE`
- 682 **Note:** The symbols above represent historical options that are no longer allowed as
 683 options, but are retained here for backwards-compatibility of applications.
- 684 • The system may support one or more options (see Section 2.1.6 (on page 32)) denoted by the
 685 following symbolic constants:
 686 — `_POSIX_ADVISORY_INFO`
 687 — `_POSIX_ASYNCHRONOUS_IO`
 688 — `_POSIX_BARRIERS`
 689 — `_POSIX_CLOCK_SELECTION`
 690 — `_POSIX_CPUTIME`
 691 — `_POSIX_FSYNC`
 692 — `_POSIX_IPV6`
 693 — `_POSIX_MAPPED_FILES`
 694 — `_POSIX_MEMLOCK`
 695 — `_POSIX_MEMLOCK_RANGE`
 696 — `_POSIX_MEMORY_PROTECTION`
 697 — `_POSIX_MESSAGE_PASSING`

698	—	_POSIX_MONOTONIC_CLOCK	
699	—	_POSIX_PRIORITIZED_IO	
700	—	_POSIX_PRIORITY_SCHEDULING	
701	—	_POSIX_RAW_SOCKETS	
702	—	_POSIX_REALTIME_SIGNALS	
703	—	_POSIX_SEMAPHORES	
704	—	_POSIX_SHARED_MEMORY_OBJECTS	
705	—	_POSIX_SPAWN	
706	—	_POSIX_SPIN_LOCKS	
707	—	_POSIX_SPORADIC_SERVER	
708	—	_POSIX_SYNCHRONIZED_IO	
709	—	_POSIX_THREAD_ATTR_STACKADDR	
710	—	_POSIX_THREAD_CPUTIME	
711	—	_POSIX_THREAD_ATTR_STACKSIZE	
712	—	_POSIX_THREAD_PRIO_INHERIT	
713	—	_POSIX_THREAD_PRIO_PROTECT	
714	—	_POSIX_THREAD_PRIORITY_SCHEDULING	
715	—	_POSIX_THREAD_PROCESS_SHARED	
716	—	_POSIX_THREAD_SAFE_FUNCTIONS	
717	—	_POSIX_THREAD_SPARADIC_SERVER	
718	—	_POSIX_THREADS	
719	—	_POSIX_TIMEOUTS	
720	—	_POSIX_TIMERS	
721	—	_POSIX_TRACE	
722	—	_POSIX_TRACE_EVENT_FILTER	
723	—	_POSIX_TRACE_INHERIT	
724	—	_POSIX_TRACE_LOG	
725	—	_POSIX_TYPED_MEMORY_OBJECTS	
726		If any of the symbolic constants <code>_POSIX_TRACE_EVENT_FILTER</code> , <code>_POSIX_TRACE_LOG</code> , or	
727		<code>_POSIX_TRACE_INHERIT</code> is defined to have a value other than <code>-1</code> , then the symbolic	
728		constant <code>_POSIX_TRACE</code> shall also be defined to have a value other than <code>-1</code> .	
729	XSI	• The system may support the XSI extensions (see Section 2.1.5.2 (on page 27)) denoted by the	
730		following symbolic constants:	
731		— <code>_XOPEN_CRYPT</code>	
732		— <code>_XOPEN_LEGACY</code>	

733 — `_XOPEN_REALTIME`

734 — `_XOPEN_REALTIME_THREADS`

735 — `_XOPEN_UNIX`

736 2.1.3.2 *POSIX Shell and Utilities*

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

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

741 • The system may support one or more options (see Section 2.1.6 (on page 32)) denoted by the
742 following symbolic constants. (The literal names below apply to the *getconf* utility.)

743 — `POSIX2_C_DEV`

744 — `POSIX2_CHAR_TERM`

745 — `POSIX2_FORT_DEV`

746 — `POSIX2_FORT_RUN`

747 — `POSIX2_LOCALEDEF`

748 — `POSIX2_PBS`

749 — `POSIX2_PBS_ACCOUNTING`

750 — `POSIX2_PBS_LOCATE`

751 — `POSIX2_PBS_MESSAGE`

752 — `POSIX2_PBS_TRACK`

753 — `POSIX2_SW_DEV`

754 — `POSIX2_UPE`

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

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

761 2.1.4 **XSI Conformance**

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

765 2.1.4.1 *XSI System Interfaces*

766 • The system shall support all the functions and headers defined in IEEE Std. 1003.1-200x as
767 part of the XSI extension denoted by the symbolic constant `_XOPEN_UNIX` and any
768 extensions marked with the XSI extension marking (see Section 1.5.1 (on page 10)).

769 • The system shall support the *mmap()*, *munmap()*, and *msync()* functions.

- 770 • The system shall support the following options defined within IEEE Std. 1003.1-200x (see
771 Section 2.1.6 (on page 32)):
- 772 — `_POSIX_FSYNC`
- 773 — `_POSIX_MAPPED_FILES`
- 774 — `_POSIX_MEMORY_PROTECTION`
- 775 — `_POSIX_THREAD_ATTR_STACKADDR`
- 776 — `_POSIX_THREAD_ATTR_STACKSIZE`
- 777 — `_POSIX_THREAD_PROCESS_SHARED`
- 778 — `_POSIX_THREAD_SAFE_FUNCTIONS`
- 779 — `_POSIX_THREADS`
- 780 • The system shall support the following Profiling Option Groups (see Section 2.1.5.1 (on page
781 25)) defined within IEEE Std. 1003.1-200x:
- 782 — `_POSIX_C_LANG_SUPPORT_R`
- 783 — `_POSIX_FILE_LOCKING`
- 784 — `_POSIX_SYSTEM_DATABASE_R`
- 785 — `_POSIX_USER_GROUPS_R`
- 786 • The system may support the following XSI Option Groups (see Section 2.1.5.2 (on page 27))
787 defined within IEEE Std. 1003.1-200x:
- 788 — `_XOPEN_CRYPT`
- 789 — `_XOPEN_LEGACY`
- 790 — `_XOPEN_REALTIME`
- 791 — `_XOPEN_REALTIME_THREADS`

792 2.1.4.2 XSI Shell and Utilities Conformance

- 793 • The system shall support all the utilities defined in the Shell and Utilities volume of
794 IEEE Std. 1003.1-200x as part of the XSI extension denoted by the XSI marking in the
795 SYNOPSIS section, and any extensions marked with the XSI extension marking (see Section
796 1.5.1 (on page 10)) within the text.
- 797 • The system shall support the User Portability Utilities option.
- 798 • The system shall support creation of locales (see Chapter 7 (on page 143)).
- 799 • The C-language Development utility *c99* shall be supported.
- 800 • The XSI Development Utilities option may be supported. It consists of the following software
801 development utilities:
- | | | | | | | | |
|-----|--------------|--------------|--------------|-------------|--|--|--|
| 802 | <i>admin</i> | <i>delta</i> | <i>rmdel</i> | <i>val</i> | | | |
| 803 | <i>cflow</i> | <i>get</i> | <i>sact</i> | <i>what</i> | | | |
| 804 | <i>ctags</i> | <i>m4</i> | <i>sccs</i> | | | | |
| 805 | <i>cxref</i> | <i>prs</i> | <i>unget</i> | | | | |
- 806 • Within the utilities that are provided, functionality marked by the codes OF, OP, PI, or UN
807 (see Section 1.5.1 (on page 10)) need not be provided.
808

809 **2.1.5 Option Groups**

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

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

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

816 **2.1.5.1 Profiling Option Groups**

817 The following Option Groups are defined to support profiling. These Option Groups allow
818 profiles to subset the System Interfaces volume of IEEE Std. 1003.1-200x by defining sets of
819 functions, denoted by the following symbolic constants:

820 `_POSIX_C_LANG_SUPPORT`: General C Library Support

821 `abs()`, `acos()`, `asctime()`, `asin()`, `atan()`, `atan2()`, `atof()`, `atoi()`, `atol()`, `bsearch()`, `calloc()`, `ceil()`,
822 `cos()`, `cosh()`, `ctime()`, `exp()`, `fabs()`, `floor()`, `fmod()`, `free()`, `frexp()`, `gmtime()`, `idexp()`, `isalnum()`,
823 `isalpha()`, `iscntrl()`, `isdigit()`, `isgraph()`, `islower()`, `isprint()`, `ispunct()`, `isspace()`, `isupper()`,
824 `isxdigit()`, `localtime()`, `log()`, `log10()`, `longjmp()`, `malloc()`, `mktime()`, `modf()`, `pow()`, `qsort()`,
825 `rand()`, `realloc()`, `setjmp()`, `sin()`, `sinh()`, `sqrt()`, `srand()`, `strcat()`, `strchr()`, `strcmp()`, `strcpy()`,
826 `strcspn()`, `strlen()`, `strncat()`, `strncmp()`, `strncpy()`, `strpbkr()`, `strrchr()`, `strspn()`, `strstr()`,
827 `strtok()`, `tan()`, `tanh()`, `tolower()`, `toupper()`

828 `_POSIX_C_LANG_SUPPORT_R`: Thread-Safe C-Language Support

829 `asctime_r()`, `ctime_r()`, `gmtime_r()`, `localtime_r()`, `readdir_r()`, `rand_r()`, `strtok_r()`

830 `_POSIX_DEVICE_IO`: Device Input and Output

831 `close()`, `clearerr()`, `getc()`, `getchar()`, `gets()`, `fclose()`, `fdopen()`, `feof()`, `ferror()`, `fflush()`, `fgetc()`,
832 `fgets()`, `fileno()`, `fopen()`, `fprintf()`, `fputc()`, `fputs()`, `fread()`, `freopen()`, `fscanf()`, `fwrite()`, `open()`,
833 `perror()`, `printf()`, `putc()`, `putchar()`, `puts()`, `read()`, `sprintf()`, `scanf()`, `sscanf()`, `setbuf()`,
834 `ungetc()`, `write()`

835 `_POSIX_DEVICE_SPECIFIC`: General Terminal Interface

836 `cfgetospeed()`, `cfsetispeed()`, `cfsetospeed()`, `ctermid()`, `isatty()`, `tcgetattr()`, `tcsetattr()`,
837 `tcsendbreak()`, `tcdrain()`, `tclflush()`, `tclflow()`, `ttyname()`

838 `_POSIX_DEVICE_SPECIFIC_R`: Thread-Safe General Terminal Interface

839 `ttyname_r()`

840 `_POSIX_FD_MGMT`: File Descriptor

841 `dup()`, `dup2()`, `fcntl()`, `fseek()`, `ftell()`, `lseek()`, `rewind()`

842 `_POSIX_FIFO`: FIFO

843 `mkfifo()`

844 `_POSIX_FILE_ATTRIBUTES`: File Attributes

845 `chmod()`, `chown()`, `umask()`

846 `_POSIX_FILE_LOCKING`: Thread-Safe Stdio Locking

847 `flockfile()`, `ftrylockfile()`, `funlockfile()`, `getc_unlocked()`, `getchar_unlocked()`, `putc_unlocked()`,
848 `putchar_unlocked()`

849 `_POSIX_FILE_SYSTEM`: File System

850 `access()`, `chdir()`, `closedir()`, `creat()`, `fpathconf()`, `fstat()`, `getcwd()`, `link()`, `mkdir()`, `opendir()`,
851 `pathconf()`, `readdir()`, `remove()`, `rename()`, `rewinddir()`, `rmdir()`, `stat()`, `tmpfile()`, `tmpnam()`,
852 `unlink()`, `utime()`

853 _POSIX_JOB_CONTROL: Job Control
854 *setpgid()*, *tcgetpgrp()*, *tcsetpgrp()*

855 _POSIX_MULTIPLE_PROCESS: Multiple Process
856 *_exit()*, *assert()*, *exit()*, *execl()*, *execle()*, *execlp()*, *execv()*, *execve()*, *execvp()*, *fork()*, *getenv()*,
857 *getpid()*, *getppid()*, *setlocale()*, *sleep()*, *times()*, *wait()*, *waitpid()*

858 _POSIX_NETWORKING: Networking
859 *accept()*, *bind()*, *connect()*, *endhostent()*, *endnetent()*, *endprotoent()*, *endservent()*, *getaddrinfo()*,
860 *gethostbyaddr()*, *gethostbyname()*, *gethostent()*, *gethostname()*, *getipnodebyaddr()*,
861 *getipnodebyname()*, *getnameinfo()*, *getnetbyaddr()*, *getnetbyname()*, *getnetent()*, *getpeername()*,
862 *getprotobyname()*, *getprotobynumber()*, *getprotoent()*, *getservbyname()*, *getservbyport()*,
863 *getservent()*, *getsockname()*, *getsockopt()*, *htonl()*, *htons()*, *if_freenameindex()*, *if_indextoname()*,
864 *if_nameindex()*, *if_nametoindex()*, *inet_addr()*, *inet_lnaof()*, *inet_makeaddr()*, *inet_netof()*,
865 *inet_network()*, *inet_ntoa()*, *listen()*, *ntohl()*, *ntohs()*, *recv()*, *recvfrom()*, *recvmsg()*, *send()*,
866 *sendmsg()*, *sendto()*, *sethostent()*, *setnetent()*, *setprotoent()*, *setservent()*, *setsockopt()*,
867 *shutdown()*, *socket()*, *socketpair()*

868 _POSIX_PIPE: Pipe
869 *pipe()*

870 _POSIX_SIGNALS: Signal
871 *abort()*, *alarm()*, *kill()*, *pause()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*,
872 *sigismember()*, *siglongjmp()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigsetjmp()*

873 _POSIX_SINGLE_PROCESS: Single Process
874 *sysconf()*, *time()*, *uname()*

875 _POSIX_SYSTEM_DATABASE: System Database
876 *getgrgid()*, *getgrnam()*, *getpwnam()*, *getpwuid()*

877 _POSIX_SYSTEM_DATABASE_R: Thread-Safe System Database
878 *getgrgid_r()*, *getgrnam_r()*, *getpwuid_r()*, *getpwnam_r()*

879 _POSIX_USER_GROUPS: User and Group
880 *geteuid()*, *getegid()*, *getgid()*, *getgroups()*, *getlogin()*, *getpgrp()*, *getuid()*, *setgid()*, *setuid()*,
881 *setuid()*

882 _POSIX_USER_GROUPS_R: Thread-Safe User and Group
883 *getlogin_r()*

884 Many of these profiling option groups provide basic system functionality that other profiling
885 option groups and options depend upon.⁴ All of the mandatory profiling option groups (listed in
886 Section 2.1.3.1 (on page 20)) shall be supported by an implementation conforming to
887 IEEE Std. 1003.1-200x. If a profile of IEEE Std. 1003.1-200x does not require an implementation to
888 provide all of the mandatory profiling option groups or does not require an implementation to
889 provide an option or profiling option group that provides features required by another option or
890 profiling option group,⁵ the profile shall specify⁶ all of the following:

- 891
- 892 4. As an example, the File System profiling option group provides underlying support for path name resolution and file creation
893 which are needed by any interface in IEEE Std. 1003.1-200x that parses a *path* argument. If a profile requires support for the
894 Device Input and Output profiling option group but does not require support for the File System profiling option group, the
895 profile must specify how path name resolution is to behave in that profile, how the *O_CREAT* flag to *open()* is to be handled (and
896 the use of the character 'a' in the *mode* argument of *fopen()* when a file name argument names a file that does not exist), and
897 specify lots of other details.
- 898 5. As an example, IEEE Std. 1003.1-200x requires that implementations claiming to support the Range Memory Locking option also
899 support the Process Memory Locking option. A profile could require that the Range Memory Locking option had to be supplied
900 without requiring that the Process Memory Locking option be supplied as long as the profile specifies everything an application
901 writer or system implementor would have to know to build an application or implementation conforming to the profile.

- 902 • Restricted or altered behavior of interfaces defined by IEEE Std. 1003.1-200x that may differ
903 on an implementation of the profile
 - 904 • Additional behaviors that may produce undefined or unspecified results
 - 905 • Additional implementation-defined behavior that implementations shall be required to
906 document in the profile’s conformance document
- 907 if any of the above is a result of the profile not providing an interface required by
908 IEEE Std. 1003.1-200x.

909 **2.1.5.2 XSI Option Groups**

910 XSI The following Option Groups are defined to support the definition of XSI conformance within
911 the System Interfaces volume of IEEE Std. 1003.1-200x:

912 **Encryption**

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

915 `crypt()`, `encrypt()`, `setkey()`

916 These functions are marked `CRYPT`.

917 Due to U.S. Government export restrictions on the decoding algorithm, implementations are
918 restricted in making these functions available. All the functions in the Encryption Option Group
919 may therefore return `[ENOSYS]` or, alternatively, `encrypt()` shall return `[ENOSYS]` for the
920 decryption operation.

921 An implementation that claims conformance to this Option Group shall set the symbolic
922 constant `_XOPEN_CRYPT` to a value other than `-1`.

923 **Realtime**

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

925 This Option Group includes a set of realtime functions drawn from options within
926 IEEE Std. 1003.1-200x (see Section 2.1.6 (on page 32)).

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

931 An implementation that claims conformance to this Option Group shall set the symbolic
932 constant `_XOPEN_REALTIME` to a value other than `-1`.

933 This Option Group consists of the set of the following options from within IEEE Std. 1003.1-200x
934 (see Section 2.1.6 (on page 32)):

935 _____

936 6. Note that the profile could just specify that any use of the features not specified by the profile would produce undefined or
937 unspecified results.

938 _POSIX_ASYNCHRONOUS_IO
 939 _POSIX_FSYNC
 940 _POSIX_MAPPED_FILES
 941 _POSIX_MEMLOCK
 942 _POSIX_MEMLOCK_RANGE
 943 _POSIX_MEMORY_PROTECTION
 944 _POSIX_MESSAGE_PASSING
 945 _POSIX_PRIORITIZED_IO
 946 _POSIX_PRIORITY_SCHEDULING
 947 _POSIX_REALTIME_SIGNALS
 948 _POSIX_SEMAPHORES
 949 _POSIX_SHARED_MEMORY_OBJECTS
 950 _POSIX_SYNCHRONIZED_IO
 951 _POSIX_TIMERS

952 If the symbolic constant `_XOPEN_REALTIME` is defined to have a value other than `-1`, then the
 953 following symbolic constants shall be defined by the implementation to have the value
 954 `200ymmL`, the date of approval of IEEE Std. 1003.1-200x:

955 _POSIX_ASYNCHRONOUS_IO
 956 _POSIX_MEMLOCK
 957 _POSIX_MEMLOCK_RANGE
 958 _POSIX_MESSAGE_PASSING
 959 _POSIX_PRIORITY_SCHEDULING
 960 _POSIX_REALTIME_SIGNALS
 961 _POSIX_SEMAPHORES
 962 _POSIX_SHARED_MEMORY_OBJECTS
 963 _POSIX_SYNCHRONIZED_IO
 964 _POSIX_TIMERS

965 The functionality associated with `_POSIX_MAPPED_FILES`, `_POSIX_MEMORY_PROTECTION`,
 966 and `_POSIX_FSYNC` is always supported on XSI-conformant systems.

967 Support of `_POSIX_PRIORITIZED_IO` on XSI-conformant systems is optional. If this
 968 functionality is supported, then `_POSIX_PRIORITIZED_IO` shall be set to a value other than `-1`.
 969 Otherwise, it shall be undefined.

970 If `_POSIX_PRIORITIZED_IO` is supported, then asynchronous I/O operations performed by
 971 `aio_read()`, `aio_write()`, and `lio_listio()` shall be submitted at a priority equal to the scheduling
 972 priority of the process minus `aiocbp->aio_reqprio`. The implementation shall also document for
 973 which files I/O prioritization is supported.

974 **Advanced Realtime**

975 An implementation that claims conformance to this Option Group shall also support the
 976 Realtime Option Group.

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

981 This Option Group consists of the set of the following options from within IEEE Std. 1003.1-200x
 982 (see Section 2.1.6 (on page 32)):

983 _POSIX_ADVISORY_INFO
 984 _POSIX_CLOCK_SELECTION
 985 _POSIX_CPUTIME
 986 _POSIX_MONOTONIC_CLOCK
 987 _POSIX_SPAWN
 988 _POSIX_SPORADIC_SERVER
 989 _POSIX_TIMEOUTS
 990 _POSIX_TYPED_MEMORY_OBJECTS

991 If the implementation supports the Advanced Realtime Option Group, then the following
 992 symbolic constants shall be defined by the implementation to have the value 200ymmL, the date
 993 of approval of IEEE Std. 1003.1-200x:

994 _POSIX_ADVISORY_INFO
 995 _POSIX_CLOCK_SELECTION
 996 _POSIX_CPUTIME
 997 _POSIX_MONOTONIC_CLOCK
 998 _POSIX_SPAWN
 999 _POSIX_SPORADIC_SERVER
 1000 _POSIX_TIMEOUTS
 1001 _POSIX_TYPED_MEMORY_OBJECTS

1002 If the symbolic constant `_POSIX_SPORADIC_SERVER` is defined, then the symbolic constant
 1003 `_POSIX_PRIORITY_SCHEDULING` shall also be defined by the implementation to have the
 1004 value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.

1005 If the symbolic constant `_POSIX_CPUTIME` is defined, then the symbolic constant
 1006 `_POSIX_TIMERS` shall also be defined by the implementation to have the value 200ymmL, the
 1007 date of approval of IEEE Std. 1003.1-200x.

1008 If the symbolic constant `_POSIX_MONOTONIC_CLOCK` is defined, then the symbolic constant
 1009 `_POSIX_TIMERS` shall also be defined by the implementation to have the value 200ymmL, the
 1010 date of approval of IEEE Std. 1003.1-200x.

1011 If the symbolic constant `_POSIX_CLOCK_SELECTION` is defined, then the symbolic constant
 1012 `_POSIX_TIMERS` shall also be defined by the implementation to have the value 200ymmL, the
 1013 date of approval of IEEE Std. 1003.1-200x.

1014 **Realtime Threads**

1015 The Realtime Threads Option Group is denoted by the symbolic constant
 1016 `_XOPEN_REALTIME_THREADS`.

1017 This Option Group consists of the set of the following options from within IEEE Std. 1003.1-200x
 1018 (see Section 2.1.6 (on page 32)):

1019 _POSIX_THREAD_PRIO_INHERIT
 1020 _POSIX_THREAD_PRIO_PROTECT
 1021 _POSIX_THREAD_PRIORITY_SCHEDULING

1022 Where applicable, whole pages are marked `REALTIME THREADS`, together with the
 1023 appropriate option margin legend for the SYNOPSIS section (see Section 1.5.1 (on page 10)).

1024 An implementation that claims conformance to this Option Group shall set
 1025 `_XOPEN_REALTIME_THREADS` to a value other than `-1`.

1026 If the symbol `_XOPEN_REALTIME_THREADS` is defined to have a value other than `-1`, then the
 1027 symbols shall also be defined by the implementation to have the value 200ymmL, the date of

1028	approval of IEEE Std. 1003.1-200x:	
1029	_POSIX_THREAD_PRIO_INHERIT	
1030	_POSIX_THREAD_PRIO_PROTECT	
1031	_POSIX_THREAD_PRIORITY_SCHEDULING	
1032	Advanced Realtime Threads	
1033	An implementation that claims conformance to this Option Group shall also support the	
1034	Realtime Threads Option Group.	
1035	Where entire functions are included in the Option Group, the NAME section is marked with	
1036	ADVANCED REALTIME THREADS. Where additional semantics have been added to existing	
1037	pages, the new material is identified by use of the appropriate margin legend for the underlying	
1038	option defined within IEEE Std. 1003.1-200x.	
1039	This Option Group consists of the set of the following options from within IEEE Std. 1003.1-200x	
1040	(see Section 2.1.6 (on page 32)):	
1041	_POSIX_BARRIERS	
1042	_POSIX_SPIN_LOCKS	
1043	_POSIX_THREAD_CPUTIME	
1044	_POSIX_THREAD_SPORADIC_SERVER	
1045	If the symbolic constant _POSIX_THREAD_SPORADIC_SERVER is defined, then the symbolic	
1046	constant _POSIX_THREAD_PRIORITY_SCHEDULING shall also be defined by the	
1047	implementation to have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.	
1048	If the symbolic constant _POSIX_THREAD_CPUTIME is defined, then the symbolic constant	
1049	_POSIX_TIMERS shall also be defined by the implementation to have the value 200ymmL, the	
1050	date of approval of IEEE Std. 1003.1-200x.	
1051	If the symbolic constant _POSIX_BARRIERS is defined, then the symbolic constants	
1052	_POSIX_THREADS and _POSIX_THREAD_SAFE_FUNCTIONS shall also be defined by the	
1053	implementation to have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.	
1054	If the symbolic constant _POSIX_SPIN_LOCKS is defined, then the symbolic constants	
1055	_POSIX_THREADS and _POSIX_THREAD_SAFE_FUNCTIONS shall also be defined by the	
1056	implementation to have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.	
1057	If the implementation supports the Advanced Realtime Threads Option Group, then the	
1058	following symbolic constants shall be defined by the implementation to have the value	
1059	200ymmL, the date of approval of IEEE Std. 1003.1-200x:	
1060	_POSIX_BARRIERS	
1061	_POSIX_SPIN_LOCKS	
1062	_POSIX_THREAD_CPUTIME	
1063	_POSIX_THREAD_SPORADIC_SERVER	

1064	Tracing
1065 1066	This Option Group includes a set of tracing functions drawn from options within IEEE Std. 1003.1-200x (see Section 2.1.6 (on page 32)).
1067 1068 1069 1070	Where entire functions are included in the Option Group, the NAME section is marked with TRACING. Where additional semantics have been added to existing pages, the new material is identified by use of the appropriate margin legend for the underlying option defined within IEEE Std. 1003.1-200x.
1071 1072	This Option Group consists of the set of the following options from within IEEE Std. 1003.1-200x (see Section 2.1.6 (on page 32)):
1073 1074 1075 1076	_POSIX_TRACE _POSIX_TRACE_EVENT_FILTER _POSIX_TRACE_LOG _POSIX_TRACE_INHERIT
1077 1078 1079	If the implementation supports the Tracing Option Group, then the following symbolic constants shall be defined by the implementation to have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x:
1080 1081 1082 1083	_POSIX_TRACE _POSIX_TRACE_EVENT_FILTER _POSIX_TRACE_LOG _POSIX_TRACE_INHERIT
1084	XSI STREAMS
1085	The XSI STREAMS Option Group is denoted by the symbolic constant <code>_XOPEN_STREAMS</code> .
1086 1087 1088	This Option Group includes functionality related to STREAMS, a uniform mechanism for implementing networking services and other character-based I/O as described in the System Interfaces volume of IEEE Std. 1003.1-200x, Section 2.6, STREAMS.
1089	It includes the following functions:
1090	<i>fattach()</i> , <i>fdetach()</i> , <i>getmsg()</i> , <i>ioctl()</i> , <i>isastream()</i> , <i>putmsg()</i> , <i>putpmsg()</i>
1091	and the <code><stropts.h></code> header.
1092 1093 1094 1095	Where applicable, whole pages are marked STREAMS, together with the appropriate option margin legend for the SYNOPSIS section (see Section 1.5.1 (on page 10)). Where additional semantics have been added to existing pages, the new material is identified by use of the appropriate margin legend for the underlying option defined within IEEE Std. 1003.1-200x.
1096	Legacy
1097	The Legacy Option Group is denoted by the symbolic constant <code>_XOPEN_LEGACY</code> .
1098 1099	The Legacy Option Group includes the functions and headers which were mandatory in previous versions of IEEE Std. 1003.1-200x but are optional in this version.
1100 1101 1102 1103	These functions and headers are retained in IEEE Std. 1003.1-200x because of their widespread use. Application writers should not rely on the existence of these functions or headers in new applications, but should follow the migration path detailed in the APPLICATION USAGE sections of the relevant pages.
1104 1105	Various factors may have contributed to the decision to mark a function or header LEGACY. In all cases, the specific reasons for the withdrawal of a function or header are documented on the

1106 relevant pages.

1107 Once a function or header is marked LEGACY, no modifications are made to the specifications
 1108 of such functions or headers other than to the APPLICATION USAGE sections of the relevant
 1109 pages.

1110 The functions and headers which form this Option Group are as follows:

1111 *bcmp()*, *bcopy()*, *bzero()*, *ecvt()*, *fcvt()*, *ftime()*, *gcvt()*, *getwd()*, *index()*, *mktemp()*, *rindex()*,
 1112 *utimes()*

1113 An implementation that claims conformance to this Option Group shall set the macro
 1114 `_XOPEN_LEGACY` to a value other than `-1`.

1115 2.1.6 Options

1116 The following symbolic constants reflect implementation options for IEEE Std. 1003.1-200x.
 1117 These symbols can be used by the application to determine which optional facilities are present
 1118 on the implementation. The *sysconf()* function defined in the System Interfaces volume of
 1119 IEEE Std. 1003.1-200x or the *getconf* utility defined in the Shell and Utilities volume of
 1120 IEEE Std. 1003.1-200x can be used to retrieve the value of each symbol on each specific
 1121 implementation.

1122 Where an option is not supported, the associated utilities, functions, or facilities need not be
 1123 present.

1124 Margin codes are defined for each option (see Section 1.5.1 (on page 10)).

1125 2.1.6.1 System Interfaces

1126 ADV `_POSIX_ADVISORY_INFO`
 1127 If this symbolic constant is defined, then the implementation supports the functions and
 1128 additional semantics in the Advisory Information option.

1129 AIO `_POSIX_ASYNCHRONOUS_IO`
 1130 If this symbolic constant is defined, then the implementation supports the functions and
 1131 additional semantics in the Asynchronous Input and Output option.

1132 BAR `_POSIX_BARRIERS`
 1133 If this symbolic constant is defined, then the implementation supports the functions and
 1134 additional semantics in the Barriers option.

1135 CS `_POSIX_CLOCK_SELECTION`
 1136 If this symbolic constant is defined, then the implementation supports the functions and
 1137 additional semantics in the Clock Selection option.

1138 CPT `_POSIX_CPUTIME`
 1139 If this symbolic constant is defined, then the implementation supports the functions and
 1140 additional semantics in the Process CPU-Time Clocks option.

1141 FSC `_POSIX_FSYNC`
 1142 If this symbolic constant is defined, then the implementation supports the functions and
 1143 additional semantics in the File Synchronization option.

1144 IP6 `_POSIX_IPV6`
 1145 If this symbol is defined, then the implementation supports the functions and additional
 1146 semantics in the IPV6 option.

1147 MF `_POSIX_MAPPED_FILES`
 1148 If this symbolic constant is defined, then the implementation supports the functions and

1149		additional semantics in the Memory Mapped Files option.
1150	ML	_POSIX_MEMLOCK
1151		If this symbolic constant is defined, then the implementation supports the functions and
1152		additional semantics in the Process Memory Locking option.
1153	MLR	_POSIX_MEMLOCK_RANGE
1154		If this symbolic constant is defined, then the implementation supports the functions and
1155		additional semantics in the Range Memory Locking option.
1156	MPR	_POSIX_MEMORY_PROTECTION
1157		If this symbolic constant is defined, then the implementation supports the functions and
1158		additional semantics in the Memory Protection option.
1159	MSG	_POSIX_MESSAGE_PASSING
1160		If this symbolic constant is defined, then the implementation supports the functions and
1161		additional semantics in the Message Passing option.
1162	MON	_POSIX_MONOTONIC_CLOCK
1163		If this symbolic constant is defined, then the implementation supports the functions and
1164		additional semantics in the Monotonic Clock option.
1165	PIO	_POSIX_PRIORITIZED_IO
1166		If this symbolic constant is defined, then the implementation supports the functions and
1167		additional semantics in the Prioritized Input and Output option.
1168	PS	_POSIX_PRIORITY_SCHEDULING
1169		If this symbolic constant is defined, then the implementation supports the functions and
1170		additional semantics in the Process Scheduling option.
1171	RTS	_POSIX_REALTIME_SIGNALS
1172		If this symbolic constant is defined, then the implementation supports the functions and
1173		additional semantics in the Realtime Signals Extension option.
1174	SEM	_POSIX_SEMAPHORES
1175		If this symbolic constant is defined, then the implementation supports the functions and
1176		additional semantics in the Semaphores option.
1177	SHM	_POSIX_SHARED_MEMORY_OBJECTS
1178		If this symbolic constant is defined, then the implementation supports the functions and
1179		additional semantics in the Shared Memory Objects option.
1180	SH	_POSIX_SHELL
1181		If this symbolic constant is defined, then the implementation supports the <i>sh</i> utility
1182		command line interpreter specified by the Shell and Utilities volume of
1183		IEEE Std. 1003.1-200x.
1184	SPN	_POSIX_SPAWN
1185		If this symbolic constant is defined, then the implementation supports the functions and
1186		additional semantics in the Spawn option.
1187	SPI	_POSIX_SPIN_LOCKS
1188		If this symbolic constant is defined, then the implementation supports the functions and
1189		additional semantics in the Spin Locks option.
1190	SS	_POSIX_SPORADIC_SERVER
1191		If this symbolic constant is defined, then the implementation supports the functions and
1192		additional semantics in the Process Sporadic Server option.

1193	SIO	_POSIX_SYNCHRONIZED_IO
1194		If this symbolic constant is defined, then the implementation supports the functions and
1195		additional semantics in the Synchronized Input and Output option.
1196	TSA	_POSIX_THREAD_ATTR_STACKADDR
1197		If this symbolic constant is defined, then the implementation supports the additional
1198		semantics in the Thread Stack Address Attribute option.
1199	TSS	_POSIX_THREAD_ATTR_STACKSIZE
1200		If this symbolic constant is defined, then the implementation supports the additional
1201		semantics in the Thread Stack Address Size option.
1202	TCT	_POSIX_THREAD_CPUTIME
1203		If this symbolic constant is defined, then the implementation supports the functions and
1204		additional semantics in the Thread CPU-Time Clocks option.
1205	TPI	_POSIX_THREAD_PRIO_INHERIT
1206		If this symbolic constant is defined, then the implementation supports the functions and
1207		additional semantics in the Threads Priority Inheritance option.
1208	TPP	_POSIX_THREAD_PRIO_PROTECT
1209		If this symbolic constant is defined, then the implementation supports the additional
1210		semantics in the Thread Priority Protection option.
1211	TPS	_POSIX_THREAD_PRIORITY_SCHEDULING
1212		If this symbolic constant is defined, then the implementation supports the functions and
1213		additional semantics in the Thread Execution Scheduling option.
1214	TSH	_POSIX_THREAD_PROCESS_SHARED
1215		If this symbolic constant is defined, then the implementation supports the additional
1216		semantics in the Thread Process-Shared Synchronization option.
1217	TSF	_POSIX_THREAD_SAFE_FUNCTIONS
1218		If this symbolic constant is defined, then the implementation supports the functions and
1219		additional semantics in the Thread-Safe Functions option.
1220	TSP	_POSIX_THREAD_SPORADIC_SERVER
1221		If this symbolic constant is defined, then the implementation supports the functions and
1222		additional semantics in the Thread Sporadic Server option.
1223	THR	_POSIX_THREADS
1224		If this symbolic constant is defined, then the implementation supports the functions and
1225		additional semantics in the Threads option.
1226	TMO	_POSIX_TIMEOUTS
1227		If this symbolic constant is defined, then the implementation supports the functions and
1228		additional semantics in the Timeouts option.
1229	TMR	_POSIX_TIMERS
1230		If this symbolic constant is defined, then the implementation supports the functions and
1231		additional semantics in the Timers option.
1232	TRC	_POSIX_TRACE
1233		If this symbolic constant is defined, then the implementation supports the functions and
1234		additional semantics in the Trace option.
1235	TEF	_POSIX_TRACE_EVENT_FILTER
1236		If this symbolic constant is defined, then the implementation supports the functions and
1237		additional semantics in the Trace Event Filter option.

1238	TRL	_POSIX_TRACE_LOG
1239		If this symbolic constant is defined, then the implementation supports the functions and
1240		additional semantics in the Trace Log option.
1241	TRI	_POSIX_TRACE_INHERIT
1242		If this symbolic constant is defined, then the implementation supports the functions and
1243		additional semantics in the Trace Inherit option.
1244	TYM	_POSIX_TYPED_MEMORY_OBJECTS
1245		If this symbolic constant is defined, then the implementation supports the functions and
1246		additional semantics in the Typed Memory Objects option.
1247	2.1.6.2	<i>Shell and Utilities</i>
1248		Each of these symbols shall be considered valid names by the implementation. Each shall be
1249		defined on the system with a value of 1 if the corresponding option is supported; otherwise, the
1250		symbol shall be undefined.
1251		The literal names shown below apply only to the <i>getconf</i> utility.
1252	CD	POSIX2_C_DEV
1253		The system supports the C-Language Development Utilities option.
1254		The utilities in the C-Language Development Utilities option are used for the development
1255		of C-language applications, including compilation or translation of C source code and
1256		complex program generators for simple lexical tasks and processing of context-free
1257		grammars.
1258		The utilities listed below may be provided by a conforming system; however, any system
1259		claiming conformance to the C-Language Development Utilities option shall provide all of
1260		the utilities listed.
1261		<i>c99</i>
1262		<i>lex</i>
1263		<i>yacc</i>
1264		POSIX2_CHAR_TERM
1265		The system supports the Terminal Characteristics option. This value need not be present on
1266		a system not supporting the User Portability Utilities option.
1267		Where applicable, the dependency is noted within the description of the utility.
1268		This option applies only to systems supporting the User Portability Utilities option. If
1269		supported, then the system supports at least one terminal type capable of all operations
1270		described in IEEE Std. 1003.1-200x; see Section 10.2 (on page 211).
1271	FD	POSIX2_FORT_DEV
1272		The system supports the FORTRAN Development Utilities option.
1273		The <i>fort77</i> FORTRAN compiler is the only utility in the FORTRAN Development Utilities
1274		option. This is used for the development of FORTRAN language applications, including
1275		compilation or translation of FORTRAN source code.
1276		The <i>fort77</i> utility may be provided by a conforming system; however, any system claiming
1277		conformance to the FORTRAN Development Utilities option shall provide the <i>fort77</i> utility.
1278	FR	POSIX2_FORT_RUN
1279		The system supports the FORTRAN Runtime Utilities option.

- 1280 The *asa* utility is the only utility in the FORTRAN Runtime Utilities option.
- 1281 The *asa* utility may be provided by a conforming system; however, any system claiming
1282 conformance to the FORTRAN Runtime Utilities option shall provide the *asa* utility.
- 1283 **POSIX2_LOCALEDEF**
- 1284 The system supports the Locale Creation Utilities option.
- 1285 If supported, the system supports the creation of locales as described in the *localedef* utility.
- 1286 The *localedef* utility may be provided by a conforming system; however, any system
1287 claiming conformance to the Locale Creation Utilities option shall provide the *localedef*
1288 utility.
- 1289 **BE** **POSIX2_PBS**
- 1290 The system supports the Batch Environment Services and Utilities option (see the Shell and
1291 Utilities volume of IEEE Std. 1003.1-200x, Chapter 3, Batch Environment Services).
- 1292 **Note:** The Batch Environment Services and Utilities option is a combination of
1293 mandatory and optional batch services and utilities. The *POSIX_PBS* symbolic
1294 constant implies the system supports all the mandatory batch services and
1295 utilities.
- 1296 **POSIX2_PBS_ACCOUNTING**
- 1297 The system supports the Batch Accounting option.
- 1298 **POSIX2_PBS_CHECKPOINT**
- 1299 The system supports the Batch Checkpoint/Restart option.
- 1300 **POSIX2_PBS_LOCATE**
- 1301 The system supports the Locate Batch Job Request option.
- 1302 **POSIX2_PBS_MESSAGE**
- 1303 The system supports the Batch Job Message Request option.
- 1304 **POSIX2_PBS_TRACK**
- 1305 The system supports the Track Batch Job Request option.
- 1306 **SD** **POSIX2_SW_DEV**
- 1307 The system supports the Software Development Utilities option.
- 1308 The utilities in the Software Development Utilities option are used for the development of
1309 applications, including compilation or translation of source code, the creation and
1310 maintenance of library archives, and the maintenance of groups of inter-dependent
1311 programs.
- 1312 The utilities listed below may be provided by the conforming system; however, any system
1313 claiming conformance to the Software Development Utilities option shall provide all of the
1314 utilities listed here.
- 1315 *ar*
- 1316 *make*
- 1317 *nm*
- 1318 *strip*
- 1319 **UP** **POSIX2_UPE**
- 1320 The system supports the User Portability Utilities option.
- 1321 The utilities in the User Portability Utilities option shall be implemented on all systems that
1322 claim conformance to this option. Certain utilities are noted as having features that cannot
1323 be implemented on all terminal types; if the *POSIX2_CHAR_TERM* option is supported, the

1324 system shall support all such features on at least one terminal type; see Section 10.2 (on
 1325 page 211).

1326 Some of the utilities are required only on systems that also support the Software
 1327 Development Utilities option, or the character-at-a-time terminal option (see Section 10.2
 1328 (on page 211)); such utilities have this noted in their DESCRIPTION sections. All of the
 1329 other utilities listed are required only on systems that claim conformance to the User
 1330 Portability Utilities option.

1331	<i>alias</i>	<i>expand</i>	<i>nm</i>	<i>unalias</i>		
1332	<i>at</i>	<i>fc</i>	<i>patch</i>	<i>unexpand</i>		
1333	<i>batch</i>	<i>fg</i>	<i>ps</i>	<i>udecode</i>		
1334	<i>bg</i>	<i>file</i>	<i>renice</i>	<i>uencode</i>		
1335	<i>crontab</i>	<i>jobs</i>	<i>split</i>	<i>vi</i>		
1336	<i>split</i>	<i>man</i>	<i>strings</i>	<i>who</i>		
1337	<i>ctags</i>	<i>mesg</i>	<i>tabs</i>	<i>write</i>		
1338	<i>df</i>	<i>more</i>	<i>talk</i>			
1339	<i>du</i>	<i>newgrp</i>	<i>time</i>			
1340	<i>ex</i>	<i>nice</i>	<i>tput</i>			

1341 2.2 Application Conformance

1342 All applications claiming conformance to IEEE Std. 1003.1-200x shall use only language-
1343 dependent services for the C programming language described in Section 2.3 (on page 40), shall
1344 use only the utilities and facilities defined in the Shell and Utilities volume of
1345 IEEE Std. 1003.1-200x, and shall fall within one of the following categories.

1346 2.2.1 Strictly Conforming POSIX Application

1347 A Strictly Conforming POSIX Application is an application that requires only the facilities
1348 described in IEEE Std. 1003.1-200x. Such an application:

- 1349 1. Shall accept any implementation behavior that results from actions it takes in areas
1350 described in IEEE Std. 1003.1-200x as *implementation-defined* or *unspecified*, or where
1351 IEEE Std. 1003.1-200x indicates that implementations may vary
- 1352 2. Shall not perform any actions that are described as producing *undefined* results
- 1353 3. For symbolic constants, shall accept any value in the range permitted by
1354 IEEE Std. 1003.1-200x, but shall not rely on any value in the range being greater than the
1355 minimums listed or being less than the maximums listed in IEEE Std. 1003.1-200x
- 1356 4. Shall not use facilities designated as *obsolescent*
- 1357 5. Is required to tolerate and permitted to adapt to the presence or absence of optional
1358 facilities whose availability is indicated by Section 2.1.3 (on page 20)
- 1359 6. For the C programming language, shall not produce any output dependent on any
1360 behavior described in the ISO C standard as *unspecified*, *undefined*, or *implementation-*
1361 *defined*, unless the System Interfaces volume of IEEE Std. 1003.1-200x specifies the behavior
- 1362 7. For the C programming language, shall not exceed any minimum implementation limit
1363 defined in the ISO C standard, unless the System Interfaces volume of
1364 IEEE Std. 1003.1-200x specifies a higher minimum implementation limit
- 1365 8. For the C programming language, shall define `_POSIX_C_SOURCE` to be 200xxxL before
1366 any header is included

1367 Within IEEE Std. 1003.1-200x, any restrictions placed upon a Conforming POSIX Application
1368 shall restrict a Strictly Conforming POSIX Application.

1369 2.2.2 Conforming POSIX Application

1370 2.2.2.1 ISO/IEC Conforming POSIX Application

1371 An ISO/IEC Conforming POSIX Application is an application that uses only the facilities
1372 described in IEEE Std. 1003.1-200x and approved Conforming Language bindings for any ISO or
1373 IEC standard. Such an application shall include a statement of conformance that documents all
1374 options and limit dependencies, and all other ISO or IEC standards used.

1375 2.2.2.2 <National Body> Conforming POSIX Application

1376 A <National Body> Conforming POSIX Application differs from an ISO/IEC Conforming
1377 POSIX Application in that it also may use specific standards of a single ISO/IEC member body
1378 referred to here as <National Body>. Such an application shall include a statement of
1379 conformance that documents all options and limit dependencies, and all other <National Body>
1380 standards used.

1381 2.2.3 Conforming POSIX Application Using Extensions

1382 A Conforming POSIX Application Using Extensions is an application that differs from a
 1383 Conforming POSIX Application only in that it uses non-standard facilities that are consistent
 1384 with IEEE Std. 1003.1-200x. Such an application shall fully document its requirements for these
 1385 extended facilities, in addition to the documentation required of a Conforming POSIX
 1386 Application. A Conforming POSIX Application Using Extensions shall be either an ISO/IEC
 1387 Conforming POSIX Application Using Extensions or a <National Body> Conforming POSIX
 1388 Application Using Extensions (see Section 2.2.2.1 (on page 38) and Section 2.2.2.2 (on page 38)).

1389 2.2.4 Strictly Conforming XSI Application

1390 A Strictly Conforming XSI Application is an application that requires only the facilities described
 1391 in IEEE Std. 1003.1-200x. Such an application:

- 1392 1. Shall accept any implementation behavior that results from actions it takes in areas
 1393 described in IEEE Std. 1003.1-200x as *implementation-defined* or *unspecified*, or where
 1394 IEEE Std. 1003.1-200x indicates that implementations may vary
- 1395 2. Shall not perform any actions that are described as producing *undefined* results
- 1396 3. For symbolic constants, shall accept any value in the range permitted by
 1397 IEEE Std. 1003.1-200x, but shall not rely on any value in the range being greater than the
 1398 minimums listed or being less than the maximums listed
- 1399 4. Shall not use facilities designated as *obsolescent*
- 1400 5. Is required to tolerate and permitted to adapt to the presence or absence of optional
 1401 facilities whose availability is indicated by Section 2.1.4 (on page 23)
- 1402 6. For the C programming language, shall not produce any output dependent on any
 1403 behavior described in the ISO C standard as *unspecified*, *undefined*, or *implementation-*
 1404 *defined*, unless the System Interfaces volume of IEEE Std. 1003.1-200x specifies the behavior
- 1405 7. For the C programming language, shall not exceed any minimum implementation limit
 1406 defined in the ISO C standard, unless the System Interfaces volume of
 1407 IEEE Std. 1003.1-200x specifies a higher minimum implementation limit
- 1408 8. For the C programming language, shall define `_XOPEN_SOURCE` to be 600 before any
 1409 header is included

1410 Within IEEE Std. 1003.1-200x, any restrictions placed upon a Conforming POSIX Application
 1411 shall restrict a Strictly Conforming XSI Application.

1412 2.2.5 Conforming XSI Application Using Extensions

1413 A Conforming XSI Application Using Extensions is an application that differs from a Strictly
 1414 Conforming XSI Application only in that it uses non-standard facilities that are consistent with
 1415 IEEE Std. 1003.1-200x. Such an application shall fully document its requirements for these
 1416 extended facilities, in addition to the documentation required of a Strictly Conforming XSI
 1417 Application.

1418 2.3 Language-Dependent Services for the C Programming Language

1419 Implementors seeking to claim conformance using the ISO C standard shall claim POSIX
1420 conformance as described in Section 2.1.3 (on page 20), C Language Binding (C Standard
1421 Language-Dependent System Support).

1422 2.4 Other Language-Related Specifications

1423 IEEE Std. 1003.1-200x is currently specified in terms of the shell command language and ISO C.
1424 Bindings to other programming languages are being developed.

1425 If conformance to IEEE Std. 1003.1-200x is claimed for implementation of any programming
1426 language, the implementation of that language shall support the use of external symbols distinct
1427 to at least 31 bytes in length in the source program text. (That is, identifiers that differ at or
1428 before the thirty-first byte shall be distinct.) If a national or international standard governing a
1429 language defines a maximum length that is less than this value, the language-defined maximum
1430 shall be supported. External symbols that differ only by case shall be distinct when the character
1431 set in use distinguishes uppercase and lowercase characters and the language permits (or
1432 requires) uppercase and lowercase characters to be distinct in external symbols.

1433

1434 3.1 Abortive Release

1435 An abrupt termination of a network connection that may result in the loss of data.

1436 3.2 Absolute Path Name

1437 A path name beginning with a single or more than two slashes; see also Section 3.268 (on page
1438 86).

1439 **Note:** Path Name Resolution is defined in detail in Section 4.9 (on page 123).

1440 3.3 Access Mode

1441 A particular form of access permitted to a file.

1442 3.4 Additional File Access Control Mechanism

1443 An implementation-defined mechanism that is layered upon the access control mechanisms
1444 defined here, but which do not grant permissions beyond those defined herein, although they
1445 may further restrict them.

1446 **Note:** File Access Permissions are defined in detail in Section 4.3 (on page 121).

1447 3.5 Address Space

1448 The memory locations that can be referenced by a process or the threads of a process.

1449 3.6 Advisory Information

1450 An interface that advises the implementation on (portable) application behavior so that it can
1451 optimize the system.

1452 **3.7 Affirmative Response**

1453 An input string that matches one of the responses acceptable to the *LC_MESSAGES* category
1454 keyword **yesexpr**, matching an extended regular expression in the current locale.

1455 **Note:** The *LC_MESSAGES* category is defined in detail in Section 7.3.6 (on page 174).

1456 **3.8 Alert**

1457 To cause the user's terminal to give some audible or visual indication that an error or some other
1458 event has occurred. When the standard output is directed to a terminal device, the method for
1459 alerting the terminal user is unspecified. When the standard output is not directed to a terminal
1460 device, the alert is accomplished by writing the <alert> character to standard output (unless the
1461 utility description indicates that the use of standard output produces undefined results in this
1462 case).

1463 **3.9 Alert Character (<alert>)**

1464 A character that in the output stream should cause a terminal to alert its user via a visual or
1465 audible notification. The <alert> character is the character designated by '\a' in the C
1466 language. It is unspecified whether this character is the exact sequence transmitted to an output
1467 device by the system to accomplish the alert function.

1468 **3.10 Alias Name**

1469 In the shell command language, a word consisting solely of underscores, digits, and alphabets
1470 from the portable character set and any of the following characters: '!', '%', ', ', '@'.

1471 Implementations may allow other characters within alias names as an extension.

1472 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 133).

1473 **3.11 Alignment**

1474 A requirement that objects of a particular type be located on storage boundaries with addresses
1475 that are particular multiples of a byte address

1476 **Note:** See also the ISO C standard, § B3.

1477 3.12 Alternate File Access Control Mechanism

1478 An implementation-defined mechanism that is independent of the access control mechanisms
1479 defined herein, and which if enabled on a file may either restrict or extend the permissions of a
1480 given user. IEEE Std. 1003.1-200x defines when such mechanisms can be enabled and when they
1481 are disabled.

1482 **Note:** File Access Permissions are defined in detail in Section 4.3 (on page 121).

1483 3.13 Alternate Signal Stack

1484 Memory associated with a thread, established upon request by the implementation for a thread,
1485 separate from the thread signal stack, in which signal handlers responding to signals sent to that
1486 thread may be executed.

1487 3.14 Ancillary Data

1488 Protocol-specific, local system-specific, or optional information. The information can be both
1489 local or end-to-end significant, header information, part of a data portion, protocol-specific, and
1490 implementation or system-specific.

1491 3.15 Angle Brackets

1492 The characters '`<`' (left-angle-bracket) and '`>`' (right-angle-bracket). When used in the phrase
1493 "enclosed in angle brackets", the symbol '`<`' immediately precedes the object to be enclosed,
1494 and '`>`' immediately follows it. When describing these characters in the portable character set,
1495 the names `<less-than-sign>` and `<greater-than-sign>` are used.

1496 3.16 Application

1497 A computer program that performs some desired function.

1498 3.17 Application Address

1499 Endpoint address of a specific application.

1500 3.18 Application Program Interface (API)

1501 The definition of syntax and semantics for providing computer system services.

1502 3.19 Appropriate Privileges

1503 An implementation-defined means of associating privileges with a process with regard to the
1504 function calls, function call options, and the commands that need special privileges. There may
1505 be zero or more such means. These means (or lack thereof) are described in the conformance
1506 document.

1507 **Note:** Function calls are defined in the System Interfaces volume of IEEE Std. 1003.1-200x,
1508 and commands are defined in the Shell and Utilities volume of IEEE Std. 1003.1-200x.

1509 3.20 Argument

1510 In the shell command language, a parameter passed to a utility as the equivalent of a single
1511 string in the *argv* array created by one of the *exec* functions. An argument is one of the options,
1512 option-arguments, or operands following the command name.

1513 **Note:** The Utility Argument Syntax is defined in detail in Section 12.1 (on page 227) and the
1514 Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.9.1.1, Command Search
1515 and Execution.

1516 In the C language, an expression in a function call expression or a sequence of preprocessing
1517 tokens in a function-like macro invocation.

1518 3.21 Arm (a Timer)

1519 To start a timer measuring the passage of time, enabling notifying a process when the specified
1520 time or time interval has passed.

1521 3.22 Assignment

1522 NEW DEF REQUIRED.

1523 **Note:** Variable Assignment is defined in detail in Section 4.16 (on page 127).

1524 3.23 Asterisk

1525 The character ' * '.

1526 3.24 Async-Cancel-Safe Function

1527 A function that may be safely invoked by an application while the asynchronous form of
1528 cancelation is enabled. No function is async-cancel-safe unless explicitly described as such.

1529 3.25 Asynchronous Events

1530 Events that occur independently of the execution of the application.

1531 3.26 Asynchronous Input and Output

1532 A functionality enhancement to allow an application process to queue data input and output
1533 commands with asynchronous notification of completion. This facility includes in its scope the
1534 requirements of supercomputer applications.

1535 3.27 Async-Signal-Safe Function

1536 A function that may be invoked, without restriction, from signal-catching functions. No function
1537 is async-signal-safe unless explicitly described as such.

1538 3.28 Asynchronously-Generated Signal

1539 A signal that is not attributable to a specific thread. Examples are signals sent via *kill()*, signals
1540 sent from the keyboard, and signals delivered to process groups. Being asynchronous is a
1541 property of how the signal was generated and not a property of the signal number. All signals
1542 may be generated asynchronously.

1543 **Note:** The *kill()* function is defined in detail in the System Interfaces volume of
1544 IEEE Std. 1003.1-200x.

1545 3.29 Asynchronous I/O Operation

1546 An I/O operation that does not of itself cause the thread requesting the I/O to be blocked from
1547 further use of the processor. |

1548 This implies that the process and the I/O operation may be running concurrently. |

1549 3.30 Asynchronous I/O Completion

1550 For an asynchronous read or write operation, when a corresponding synchronous read or write
1551 would have completed and when any associated status fields have been updated. |

1552 3.31 Authentication

1553 The process of validating a user or process to verify that the user or process is not a counterfeit.

1554 3.32 Authorization

1555 The process of verifying that a user or process has permission to use a resource in the manner
1556 requested.

1557 To ensure security, the user or process would also need to be authenticated before granting
1558 access.

1559 3.33 Background Job

1560 See *Background Process Group* in Section 3.35.

1561 3.34 Background Process

1562 A process that is a member of a background process group.

1563 3.35 Background Process Group (or Background Job)

1564 Any process group, other than a foreground process group, that is a member of a session that
1565 has established a connection with a controlling terminal.

1566 3.36 Backquote

1567 The character ' ` ', also known as a *grave accent*.

1568 3.37 Backslash

1569 The character ' \ ', also known as a *reverse solidus*.

1570 3.38 Backspace Character (<backspace>)

1571 A character that, in the output stream, should cause printing (or displaying) to occur one column
1572 position previous to the position about to be printed. If the position about to be printed is at the
1573 beginning of the current line, the behavior is unspecified. The <backspace> character is the
1574 character designated by ' \b ' in the C language. It is unspecified whether this character is the
1575 exact sequence transmitted to an output device by the system to accomplish the backspace
1576 function. The <backspace> character defined here is not necessarily the ERASE special character.

1577 **Note:** Special Characters are defined in detail in Section 11.1.9 (on page 217).

1578 3.39 Barrier

1579 A synchronization object that allows multiple threads to synchronize at a particular point in
1580 their execution.

1581 3.40 Base Character

1582 One of the set of characters defined in the Latin alphabet. In Western European languages other
1583 than English, these characters are commonly used with diacritical marks (accents, cedilla, and so
1584 on) to extend the range of characters in an alphabet.

1585 3.41 Basename

1586 The final, or only, file name in a path name.

1587 3.42 Basic Regular Expression (BRE)

1588 A regular expression (see Section 3.318 (on page 95)) used by the majority of utilities that select
1589 strings from a set of character strings.

1590 **Note:** Basic Regular Expressions are described in detail in Section 9.3 (on page 198).

1591 3.43 Batch Access List

1592 A list of user IDs and group IDs of those users and groups authorized to place batch jobs in a
1593 batch queue.

1594 A batch access list is associated with a batch queue. A batch server uses the batch access list of a
1595 batch queue as one of the criteria in deciding to put a batch job in a batch queue.

1596 3.44 Batch Administrator

1597 A person who is authorized to use all restricted batch services.

1598 3.45 Batch Client

1599 A computational entity that utilizes batch services by making requests of batch servers.

1600 Batch clients often provide the means by which users access batch services, although a batch
1601 server may act as a batch client by virtue of making requests of another batch server.

1602 3.46 Batch Destination

1603 The batch server in a batch system to which a batch job should be sent for processing.

1604 Acceptance of a batch job at a batch destination is the responsibility of a receiving batch server.
1605 A batch destination may consist of a batch server-specific portion, a network-wide portion, or
1606 both. The batch server-specific portion is referred to as the *batch queue*. The network-wide
1607 portion is referred to as a *batch server name*.

1608 3.47 Batch Destination Identifier

1609 A string that identifies a specific batch destination.

1610 A string of characters in the portable character set used to specify a particular batch destination.

1611 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 133).

1612 3.48 Batch Directive

1613 A line from a file that is interpreted by the batch server. The line is usually in the form of a
1614 comment and is an additional means of passing options to the *qsub* utility.

1615 **Note:** The *qsub* utility is defined in detail in the Shell and Utilities volume of
1616 IEEE Std. 1003.1-200x.

1617 3.49 Batch Job

1618 A set of computational tasks for a computing system.

1619 Batch jobs are managed by batch servers.

1620 Once created, a batch job may be executing or pending execution. A batch job that is executing
1621 has an associated session leader (a process) that initiates and monitors the computational tasks
1622 of the batch job.

1623 3.50 Batch Job Attribute

1624 A named data type whose value affects the processing of a batch job.

1625 The values of the attributes of a batch job affect the processing of that job by the batch server
1626 that manages the batch job.

1627 The attributes defined for a batch job are called the batch job attributes.

1628 3.51 Batch Job Identifier

1629 A unique name for a batch job. A name that is unique among all other batch job identifiers in a
1630 batch system and that identifies the batch server to which the batch job was originally
1631 submitted.

1632 3.52 Batch Job Name

1633 A label that is an attribute of a batch job. The batch job name is not necessarily unique.

1634 3.53 Batch Job Owner

1635 The *username@hostname* of the user submitting the batch job, where *username* is a user name (see
1636 also Section 3.428 (on page 115)) and *hostname* is a network host name.

1637 3.54 Batch Job Priority

1638 An attribute used in selecting a batch job for execution.

1639 A value specified by the user that may be used by an implementation to determine the order in
1640 which batch jobs are selected to be executed. Job priority has a numeric value in the range -1 024
1641 to 1 023.

1642 **Note:** The batch job priority is not the execution priority (nice value) of the batch job.

1643 3.55 Batch Job State

1644 An attribute of a batch job.

1645 The state of a batch job determines the types of requests that the batch server that manages the
1646 batch job can accept for the batch job. Valid states include QUEUED, RUNNING, HELD,
1647 WAITING, EXITING, and TRANSITING.

1648 3.56 Batch Name Service

1649 A service that assigns batch names that are unique within the batch name space, and that can
1650 translate a unique batch name into the location of the named batch entity.

1651 3.57 Batch Name Space

1652 The environment within which a batch name is known to be unique.

1653 3.58 Batch Node

1654 A host containing part or all of a batch system.

1655 A batch node is a host meeting at least one of the following conditions:

- 1656 • Capable of executing a batch client
- 1657 • Contains a routing batch queue
- 1658 • Contains an execution batch queue

1659 3.59 Batch Operator

1660 A person who is authorized to use some, but not all, restricted batch services. |

1661 3.60 Batch Queue

1662 A manageable object that represents a set of batch jobs and is managed by a single batch server. |

1663 **Note:** Each batch job managed by a batch server is a member of a single batch queue
1664 managed by that server.

1665 Such a set of batch jobs is called a batch queue largely for historical reasons. Jobs are
1666 selected from the batch queue for execution based on attributes such as priority,
1667 resource requirements, and hold conditions.

1668 Two types of batch queue are described in IEEE Std. 1003.1-200x: *routing batch queues*
1669 and *execution batch queues*.

1670 3.61 Batch Queue Attribute

1671 A named data type whose value affects the processing of all batch jobs that are members of the
1672 batch queue. |

1673 A batch queue has attributes that affect the processing of batch jobs that are members of the
1674 batch queue.

1675 The attributes defined for a batch queue are called the batch batch queue attributes. |

1676 3.62 Batch Queue Position

1677 The place a batch job occupies in a batch queue. |

1678 This place is relative to other batch jobs in the batch queue and defined in part by submission
1679 time and its priority; see also Section 3.63. |

1680 3.63 Batch Queue Priority

1681 The maximum job priority allowed for any batch job in a given batch queue. |

1682 The batch queue priority is set and may be changed by users with appropriate privilege. The
1683 priority is bounded in an implementation-defined manner. |

1684 3.64 Batch Rerunability

1685 An attribute of a batch job.

1686 If a batch job may be rerun from the beginning after an abnormal termination without affecting
1687 the validity of the results, the batch job is said to be rerunable.

1688 3.65 Batch Restart

1689 Resume the processing of a batch job from the point of the last checkpoint. Typically, this is done
1690 if the batch job has been interrupted because of a system failure.

1691 3.66 Batch Server

1692 A computational entity that provides batch services.

1693 3.67 Batch Server Name

1694 A string that identifies a specific server in a network.

1695 A string of characters in the portable character set used to specify a particular server in a
1696 network.

1697 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 133).

1698 3.68 Batch Service

1699 Computational and organizational services performed by a batch system on behalf of batch jobs.

1700 Batch services are of two types: *requested* and *deferred*.

1701 **Note:** Batch Services are listed in the Shell and Utilities volume of IEEE Std. 1003.1-200x,
1702 Table 3-5, Batch Services Summary.

1703 3.69 Batch Service Request

1704 A solicitation of services from a batch client to a batch server.

1705 A batch service request may entail the exchange of any number of messages between the batch
1706 client and the batch server.

1707 When naming specific types of service requests, the term request is qualified by the type of
1708 request, as in *Queue Batch Job Request* and *Delete Batch Job Request*.

1709 3.70 Batch Submission

1710 The process by which a batch client requests that a batch server create a batch job via a *Queue Job*
1711 *Request* to perform a specified computational task.

1712 3.71 Batch System

1713 A collection of one or more batch servers.

1714 3.72 Batch Target User

1715 The name of a user on the batch destination batch server.

1716 The target user is the user name under whose account the batch job is to execute on the
1717 destination batch server.

1718 3.73 Batch User

1719 A person who is authorized to make use of batch services.

1720 3.74 Bind

1721 Assign a network address to an endpoint.

1722 3.75 Blank Character (<blank>)

1723 One of the characters that belong to the **blank** character class as defined via the *LC_CTYPE*
1724 category in the current locale. In the POSIX locale, a <blank> character is either a <tab> or a
1725 <space> character.

1726 3.76 Blank Line

1727 A line consisting solely of zero or more <blank> characters terminated by a <newline> character;
1728 see also Section 3.146 (on page 66).

1729 3.77 Blocked Process (or Thread)

1730 A process (or thread) that is waiting for some condition (other than the availability of a
1731 processor) to be satisfied before it can continue execution.

1732 3.78 Blocking

1733 Executing with O_NONBLOCK not set; see also Section 3.242 (on page 82).

1734 3.79 Block-Mode Terminal

1735 A terminal device operating in a mode incapable of the character-at-a-time input and output
1736 operations described by some of the standard utilities.

1737 **Note:** Output Devices and Terminal Types are defined in detail in Section 10.2 (on page
1738 211).

1739 3.80 Block Special File

1740 A file that refers to a device. A block special file is normally distinguished from a character
1741 special file by providing access to the device in a manner such that the hardware characteristics
1742 of the device are not visible.

1743 3.81 Braces

1744 The characters '{' (left brace) and '}' (right brace), also known as *curly braces*. When used in
1745 the phrase “enclosed in (curly) braces” the symbol '{' immediately precedes the object to be
1746 enclosed, and '}' immediately follows it. When describing these characters in the portable
1747 character set, the names <left-brace> and <right-brace> are used.

1748 3.82 Brackets

1749 The characters '[' (left-bracket) and ']' (right-bracket), also known as *square brackets*. When
1750 used in the phrase “enclosed in (square) brackets” the symbol '[' immediately precedes the
1751 object to be enclosed, and ']' immediately follows it. When describing these characters in the
1752 portable character set, the names <left-square-bracket> and <right-square-bracket> are used.

1753 3.83 Break Value

1754 The address at which dynamic memory allocation starts.

1755 3.84 Broadcast

1756 The transfer of data from one endpoint to several endpoints, as described in RFC 919 and
1757 RFC 922.

1758 3.85 Built-In Utility (or Built-In)

1759 A utility implemented within a shell. The utilities referred to as *special built-ins* have special
1760 qualities. Unless qualified, the term built-in includes the special built-in utilities. *Regular built-ins*
1761 are not required to be actually built into the shell on the implementation, but they do have
1762 special command-search qualities.

1763 **Note:** Special Built-In Utilities are defined in detail in the Shell and Utilities volume of
1764 IEEE Std. 1003.1-200x, Section 2.15, Special Built-In Utilities.

1765 Regular Built-In Utilities are defined in detail in the Shell and Utilities volume of
1766 IEEE Std. 1003.1-200x, Section 2.9.1.1, Command Search and Execution.

1767 3.86 Byte

1768 An individually addressable unit of data storage that is equal to or larger than an octet, used to
1769 store a character or a portion of a character; see also Section 3.89 (on page 56). A byte is
1770 composed of a contiguous sequence of bits, the number of which is implementation-defined. The
1771 least significant bit is called the *low-order* bit; the most significant is called the *high-order* bit.

1772 **Note:** The definition of *byte* is actually from the ISO C standard. It has been reworded
1773 slightly to clarify its intent without introducing the ISO C standard terminology
1774 “basic execution character set”, which is inapplicable to IEEE Std. 1003.1-200x. It
1775 deviates intentionally from the usage of *byte* in some international standards, where
1776 it is used as a synonym for *octet* (always eight bits). A byte may be larger than eight
1777 bits so that it can be an integral portion of larger data objects that are not evenly
1778 divisible by eight bits (such as a 36-bit word that contains four 9-bit bytes).

1779 3.87 Byte Input/Output Functions

1780 The functions that perform byte-oriented input from streams or byte-oriented output to streams:
1781 *fgetc()*, *fgets()*, *fprintf()*, *fputc()*, *fputs()*, *fread()*, *fscanf()*, *fwrite()*, *getc()*, *getchar()*, *gets()*, *printf()*,
1782 *putc()*, *putchar()*, *puts()*, *scanf()*, *ungetc()*, *vfprintf()*, and *vprintf()*.

1783 **Note:** Functions are defined in detail in the System Interfaces volume of
1784 IEEE Std. 1003.1-200x.

1785 3.88 Carriage-Return Character (<carriage-return>)

1786 A character that in the output stream indicates that printing should start at the beginning of the
1787 same physical line in which the <carriage-return> character occurred. The <carriage-return>
1788 character is the character designated by '`\r`' in the C language. It is unspecified whether this
1789 character is the exact sequence transmitted to an output device by the system to accomplish the
1790 movement to the beginning of the line.

1791 3.89 Character

1792 A sequence of one or more bytes representing a single graphic symbol or control code.

1793 **Note:** This term corresponds to the ISO C standard term multi-byte character, where a
1794 single-byte character is a special case of a multi-byte character. Unlike the usage in
1795 the ISO C standard, *character* here has no necessary relationship with storage space,
1796 and *byte* is used when storage space is discussed.

1797 See the definition of the Portable Character Set in Section 6.1 (on page 133) for a
1798 further explanation of the graphical representations of characters, or *glyphs*, as
1799 opposed to character encodings.

1800 3.90 Character Array

1801 An array of elements of type **char**.

1802 3.91 Character Class

1803 A named set of characters sharing an attribute associated with the name of the class. The classes
1804 and the characters that they contain are dependent on the value of the *LC_CTYPE* category in the
1805 current locale.

1806 **Note:** The *LC_CTYPE* category is defined in detail in Section 7.3.1 (on page 147).

1807 3.92 Character Set

1808 A finite set of different characters used for the representation, organization, or control of data.

1809 3.93 Character Special File

1810 A file that refers to a device. One specific type of character special file is a terminal device file.

1811 **Note:** The General Terminal Interface is defined in detail in Chapter 11 (on page 213).

1812 3.94 Character String

1813 A contiguous sequence of characters terminated by and including the first null byte.

1814 3.95 Child Process

1815 A new process created (by *fork()* or *spawn()*) by a given process. A child process remains the
1816 child of the creating process as long as both processes continue to exist.

1817 **Note:** The *fork()* and *spawn()* functions are defined in detail in the System Interfaces
1818 volume of IEEE Std. 1003.1-200x.

1819 3.96 Circumflex

1820 The character '^'.

1821 3.97 Clock

1822 A software or hardware object that can be used to measure the apparent or actual passage of
1823 time.

1824 The current value of the time measured by a clock can be queried and, possibly, set to a value
1825 within the legal range of the clock.

1826 3.98 Clock Jump

1827 The difference between two successive distinct values of a clock, as observed from the
1828 application via one of the "get time" operations.

1829 3.99 Clock Tick

1830 An interval of time; an implementation-defined number of these occur each second. Clock ticks
1831 are one of the units that may be used to express a value found in type `clock_t`.

1832 3.100 Coded Character Set

1833 A set of unambiguous rules that establishes a character set and the one-to-one relationship
1834 between each character of the set and its bit representation.

1835 3.101 Codeset

1836 The result of applying rules that map a numeric code value to each element of a character set. An
1837 element of a character set may be related to more than one numeric code value but the reverse is
1838 not true. However, for state-dependent encodings the relationship between numeric code values
1839 to elements of a character set may be further controlled by state information. The character set
1840 may contain fewer elements than the total number of possible numeric code values; that is, some
1841 code values may be unassigned.

1842 **Note:** Character Encoding is defined in detail in Section 6.2 (on page 136).

1843 3.102 Collating Element

1844 The smallest entity used to determine the logical ordering of character or wide-character strings;
1845 see also Section 3.105 (on page 59). A collating element consists of either a single character, or
1846 two or more characters collating as a single entity. The value of the `LC_COLLATE` category in the
1847 current locale determines the current set of collating elements.

1848 3.103 Collating Element Order

1849 The relative order of collating elements as determined by the setting of the `LC_COLLATE`
1850 category in the current locale.

1851 The collating element order is used in range expressions in REs and is determined by the order in
1852 which collating elements are specified between `order_start` and `order_end` keywords in the
1853 `LC_COLLATE` category.

1854 3.104 Collation

1855 The logical ordering of character or wide-character strings according to defined precedence
1856 rules. These rules identify a collation sequence between the collating elements, and such
1857 additional rules that can be used to order strings consisting of multiple collating elements.

1858 3.105 Collation Sequence

1859 The relative order of collating elements as determined by the setting of the *LC_COLLATE*
1860 category in the current locale. The collation sequence is used for sorting and is determined from
1861 the collating weights assigned to each collating element. In the absence of weights, the collation
1862 sequence is also the collating element order.

1863 Multi-level sorting is accomplished by assigning elements one or more collation weights, up to
1864 the limit {*COLL_WEIGHTS_MAX*}. On each level, elements may be given the same weight (at
1865 the primary level, called an equivalence class; see also Section 3.152 (on page 66)) or be omitted
1866 from the sequence. Strings that collate equal using the first assigned weight (primary ordering)
1867 are then compared using the next assigned weight (secondary ordering), and so on.

1868 **Note:** {*COLL_WEIGHTS_MAX*} is defined in detail in `<limits.h>`.

1869 3.106 Column Position

1870 A unit of horizontal measure related to characters in a line.

1871 It is assumed that each character in a character set has an intrinsic column width independent of
1872 any output device. Each printable character in the portable character set has a column width of
1873 one. The standard utilities, when used as described in IEEE Std. 1003.1-200x, assume that all
1874 characters have integral column widths. The column width of a character is not necessarily
1875 related to the internal representation of the character (numbers of bits or bytes).

1876 The column position of a character in a line is defined as one plus the sum of the column widths
1877 of the preceding characters in the line. Column positions are numbered starting from 1.

1878 3.107 Command

1879 A directive to the shell to perform a particular task.

1880 **Note:** Shell Commands are defined in detail in the Shell and Utilities volume of
1881 IEEE Std. 1003.1-200x, Section 2.9, Shell Commands. |

1882 3.108 Command Language Interpreter

1883 An interface that interprets sequences of text input as commands. It may operate on an input
1884 stream or it may interactively prompt and read commands from a terminal. It is possible for
1885 applications to invoke utilities through a number of interfaces, which are collectively considered
1886 to act as command interpreters. The most obvious of these are the *sh* utility and the *system()*
1887 function, although *popen()* and the various forms of *exec* may also be considered to behave as
1888 interpreters.

1889 **Note:** The *sh* utility is defined in detail in the Shell and Utilities volume of
1890 IEEE Std. 1003.1-200x.

1891 The *system()*, *popen()*, and *exec* functions are defined in detail in the System Interfaces
1892 volume of IEEE Std. 1003.1-200x.

1893 3.109 Composite Graphic Symbol

1894 A graphic symbol consisting of a combination of two or more other graphic symbols in a single
1895 character position, such as a diacritical mark and a base character.

1896 3.110 Condition Variable

1897 A synchronization object which allows a thread to suspend execution, repeatedly, until some
1898 associated predicate becomes true. A thread whose execution is suspended on a condition
1899 variable is said to be blocked on the condition variable.

1900 3.111 Connection

1901 An association established between two or more endpoints for the transfer of data

1902 3.112 Connection Mode

1903 The transfer of data in the context of a connection; see also Section 3.113.

1904 3.113 Connectionless Mode

1905 The transfer of data other than in the context of a connection; see also Section 3.112 and Section
1906 3.126 (on page 62).

1907 3.114 Control Character

1908 A character, other than a graphic character, that affects the recording, processing, transmission,
1909 or interpretation of text.

1910 3.115 Control Operator

1911 In the shell command language, a token that performs a control function. It is one of the
1912 following symbols:

1913 & && () ; ;; newline | ||

1914 The end-of-input indicator used internally by the shell is also considered a control operator.

1915 **Note:** Token Recognition is defined in detail in the Shell and Utilities volume of
1916 IEEE Std. 1003.1-200x, Section 2.3, Token Recognition.

1917 3.116 Controlling Process

1918 The session leader that established the connection to the controlling terminal. If the terminal
1919 subsequently ceases to be a controlling terminal for this session, the session leader ceases to be
1920 the controlling process.

1921 3.117 Controlling Terminal

1922 A terminal that is associated with a session. Each session may have at most one controlling
1923 terminal associated with it, and a controlling terminal is associated with exactly one session.
1924 Certain input sequences from the controlling terminal cause signals to be sent to all processes in
1925 the process group associated with the controlling terminal.

1926 **Note:** The General Terminal Interface is defined in detail in Chapter 11 (on page 213).

1927 3.118 Conversion Descriptor

1928 A per-process unique value used to identify an open codeset conversion.

1929 3.119 Core File

1930 A file of unspecified format that may be generated when a process terminates abnormally.

1931 3.120 CPU Time (Execution Time)

1932 The time spent executing a process or thread, including the time spent executing system services
1933 on behalf of that process or thread. If the Threads option is supported, then the value of the
1934 CPU-time clock for a process is implementation-defined. With this definition the sum of all the
1935 execution times of all the threads in a process might not equal the process execution time, even
1936 in a single-threaded process, because implementations may differ in how they account for time
1937 during context switches or for other reasons.

1938 3.121 CPU-Time Clock

1939 A clock that measures the execution time of a particular process or thread.

1940 3.122 CPU-Time Timer

1941 A timer attached to a CPU-time clock.

1942 3.123 Current Job

1943 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities. There
1944 is at most one current job; see also Section 3.205 (on page 76).

1945 3.124 Current Working Directory

1946 See *Working Directory* in Section 3.438 (on page 117).

1947 3.125 Cursor Position

1948 The line and column position on the screen denoted by the terminal's cursor.

1949 3.126 Datagram

1950 A unit of data transferred from one endpoint to another in connectionless mode service.

1951 **3.127 Data Segment**

1952 Memory associated with a process, that can contain dynamically allocated data. |

1953 **3.128 Deferred Batch Service**

1954 A service that is performed as a result of events that are asynchronous with respect to requests. |

1955 **Note:** Once a batch job has been created, it is subject to deferred services.

1956 **3.129 Device**

1957 A computer peripheral or an object that appears to the application as such.

1958 **3.130 Device ID**

1959 A non-negative integer used to identify a device. |

1960 **3.131 Directory**

1961 A file that contains directory entries. No two directory entries in the same directory have the |
1962 same name. |

1963 **3.132 Directory Entry (or Link)**

1964 An object that associates a file name with a file. Several directory entries can associate names
1965 with the same file.

1966 **3.133 Directory Stream**

1967 A sequence of all the directory entries in a particular directory. An open directory stream may be
1968 implemented using a file descriptor.

1969 **3.134 Disarm (a Timer)**

1970 To stop a timer from measuring the passage of time, disabling any future process notifications
 1971 (until the timer is armed again).

1972 **3.135 Display**

1973 To output to the user's terminal. If the output is not directed to a terminal, the results are
 1974 undefined.

1975 **3.136 Dollar Sign**

1976 The character '\$'.

1977 **3.137 Dot**

1978 In the context of naming files, the file name consisting of a single dot character ('.').

1979 **Note:** In the context of shell special built-in utilities, see *dot* in the Shell and Utilities volume
 1980 of IEEE Std. 1003.1-200x, Section 2.15, Special Built-In Utilities.

1981 Path Name Resolution is defined in detail in Section 4.9 (on page 123).

1982 **3.138 Dot-Dot**

1983 The file name consisting solely of two dot characters ("..").

1984 **Note:** Path Name Resolution is defined in detail in Section 4.9 (on page 123).

1985 **3.139 Double-Quote**

1986 The character '"', also known as *quotation-mark*.

1987 **Note:** The *double* adjective in this term refers to the two strokes in the character glyph.
 1988 IEEE Std. 1003.1-200x never uses the term double-quote to refer to two apostrophes
 1989 or quotation marks.

1990 3.140 Downshifting

1991 The conversion of an uppercase character that has a single-character lowercase representation
1992 into this lowercase representation.

1993 3.141 Driver

1994 A module that controls data transferred to and received from devices.

1995 **Note:** Drivers are traditionally written to be a part of the system implementation, although
1996 they are frequently written separately from the writing of the implementation. A
1997 driver may contain processor-specific code, and therefore be non-portable.

1998 3.142 Effective Group ID

1999 An attribute of a process that is used in determining various permissions, including file access
2000 permissions; see also Section 3.190 (on page 73).

2001 3.143 Effective User ID

2002 An attribute of a process that is used in determining various permissions, including file access
2003 permissions; see also Section 3.427 (on page 115).

2004 3.144 Eight-Bit Transparency

2005 The ability of a software component to process 8-bit characters without modifying or utilizing
2006 any part of the character in a way that is inconsistent with the rules of the current coded
2007 character set.

2008 3.145 Empty Directory

2009 A directory that contains, at most, directory entries for dot and dot-dot, and has exactly one link
2010 to it in dot-dot. No other links to the directory may exist. It is unspecified whether an
2011 implementation can ever consider the root directory to be empty.

2012 3.146 Empty Line

2013 A line consisting of only a <newline> character; see also Section 3.76 (on page 53).

2014 3.147 Empty String (or Null String)

2015 A string whose first byte is a null byte.

2016 3.148 Empty Wide-Character String

2017 A wide-character string whose first element is a null wide-character code.

2018 3.149 Encoding Rule

2019 The rules used to convert between wide-character codes and multi-byte character codes.

2020 **Note:** Stream Orientation and Encoding Rules are defined in detail in the System Interfaces
2021 volume of IEEE Std. 1003.1-200x, Section 2.5.2, Stream Orientation and Encoding
2022 Rules.

2023 3.150 Entire Regular Expression

2024 The concatenated set of one or more BREs or EREs that make up the pattern specified for string
2025 selection.

2026 **Note:** Regular Expressions are defined in detail in Chapter 9 (on page 195).

2027 3.151 Epoch

2028 The time zero hours, zero minutes, zero seconds, on January 1, 1970 Coordinated Universal
2029 Time.

2030 **Note:** See also Seconds Since the Epoch defined in Section 4.12 (on page 125).

2031 3.152 Equivalence Class

2032 A set of collating elements with the same primary collation weight.

2033 Elements in an equivalence class are typically elements that naturally group together, such as all
2034 accented letters based on the same base letter.

2035 The collation order of elements within an equivalence class is determined by the weights
2036 assigned on any subsequent levels after the primary weight.

2037 3.153 Era

2038 An alternative method for counting and displaying years.

2039 **Note:** The *LC_TIME* category is defined in detail in Section 7.3.5 (on page 168).

2040 3.154 Event Management

2041 The mechanism that enables applications to register for and be made aware of external events
2042 such as data becoming available for reading.

2043 3.155 Executable File

2044 A regular file acceptable as a new process image file by the equivalent of the *exec* family of
2045 functions, and thus usable as one form of a utility. The standard utilities described as compilers
2046 can produce executable files, but other unspecified methods of producing executable files may
2047 also be provided. The internal format of an executable file is unspecified, but a conforming
2048 application cannot assume an executable file is a text file.

2049 3.156 Execute

2050 In the Shell and Utilities volume of IEEE Std. 1003.1-200x, to perform command search and
2051 execution actions; see also Section 3.202 (on page 75).

2052 **Note:** Command Search and Execution is defined in detail in the Shell and Utilities volume
2053 of IEEE Std. 1003.1-200x, Section 2.9.1.1, Command Search and Execution.

2054 3.157 Execution Time

2055 See *CPU Time* in Section 3.120 (on page 62).

2056 3.158 Execution Time Monitoring

2057 A set of execution time monitoring primitives that allow online measuring of thread and process
2058 execution times.

2059 3.159 Expand

2060 In the shell command language, when not qualified, the act of applying word expansions.

2061 **Note:** Work Expansions are defined in detail in the Shell and Utilities volume of
2062 IEEE Std. 1003.1-200x, Section 2.6, Word Expansions.

2063 3.160 Extended Regular Expression (ERE)

2064 A regular expression (see also Section 3.318 (on page 95)) that is an alternative to the Basic
2065 Regular Expression using a more extensive syntax, occasionally used by some utilities.

2066 **Note:** Extended Regular Expressions are described in detail in Section 9.4 (on page 203).

2067 3.161 Extended Security Controls

2068 Implementation-defined security controls allowed by the file access permission and appropriate
2069 privilege (see also Section 3.19 (on page 44)) mechanisms, through which an implementation can
2070 support different security policies from those described in IEEE Std. 1003.1-200x.

2071 **Note:** See also Extended Security Controls defined in Section 4.2 (on page 121).

2072 File Access Permissions are defined in detail in Section 4.3 (on page 121).

2073 3.162 Feature Test Macro

2074 A macro used to determine whether a particular set of features is included from a header.

2075 **Note:** See also the System Interfaces volume of IEEE Std. 1003.1-200x, Section 2.2, The
2076 Compilation Environment.

2077 3.163 Field

2078 In the shell command language, a unit of text that is the result of parameter expansion,
2079 arithmetic expansion, command substitution, or field splitting. During command processing, the
2080 resulting fields are used as the command name and its arguments.

2081 **Note:** Parameter Expansion is defined in detail in the Shell and Utilities volume of
2082 IEEE Std. 1003.1-200x, Section 2.6.2, Parameter Expansion.

2083 Arithmetic Expansion is defined in detail in the Shell and Utilities volume of
2084 IEEE Std. 1003.1-200x, Section 2.6.4, Arithmetic Expansion.

2085 Command Substitution is defined in detail in the Shell and Utilities volume of
2086 IEEE Std. 1003.1-200x, Section 2.6.3, Command Substitution.

2087 Field Splitting is defined in detail in the Shell and Utilities volume of
2088 IEEE Std. 1003.1-200x, Section 2.6.5, Field Splitting.

2089 For further information on command processing, see the Shell and Utilities volume of
2090 IEEE Std. 1003.1-200x, Section 2.9.1, Simple Commands.

2091 3.164 FIFO Special File (or FIFO)

2092 A type of file with the property that data written to such a file is read on a first-in-first-out basis.

2093 **Note:** Other characteristics of FIFOs are described in the System Interfaces volume of
2094 IEEE Std. 1003.1-200x, *lseek()*, *open()*, *read()*, and *write()*.

2095 3.165 File

2096 An object that can be written to, or read from, or both. A file has certain attributes, including
2097 access permissions and type. File types include regular file, character special file, block special
2098 file, FIFO special file, symbolic link, socket, and directory. Other types of files may be supported
2099 by the implementation.

2100 3.166 File Description

2101 See *Open File Description* in Section 3.255 (on page 84).

2102 3.167 File Descriptor

2103 A per-process unique, non-negative integer used to identify an open file for the purpose of file
2104 access. The value of a file descriptor is from zero to {OPEN_MAX}. A process can have no more
2105 than {OPEN_MAX} file descriptors open simultaneously. File descriptors may also be used to
2106 implement message catalog descriptors and directory streams; see also Section 3.255 (on page
2107 84).

2108 **Note:** {OPEN_MAX} is defined in detail in <limits.h>.

2109 3.168 File Group Class

2110 The property of a file indicating access permissions for a process related to the group
2111 identification of a process. A process is in the file group class of a file if the process is not in the
2112 file owner class and if the effective group ID or one of the supplementary group IDs of the
2113 process matches the group ID associated with the file. Other members of the class may be
2114 implementation-defined.

2115 3.169 File Mode

2116 An object containing the *file mode bits* and file type of a file.

2117 **Note:** File mode bits and file types are defined in detail in `<sys/stat.h>`.

2118 3.170 File Mode Bits

2119 A file's file permission bits, set-user-ID-on-execution bit (S_ISUID), and set-group-ID-on-
2120 execution bit (S_ISGID).

2121 **Note:** File Mode Bits are defined in detail in `<sys/stat.h>`.

2122 3.171 File Name

2123 A name consisting of 1 to {NAME_MAX} bytes used to name a file. The characters composing
2124 the name may be selected from the set of all character values excluding the slash character and
2125 the null byte. The file names dot and dot-dot have special meaning. A file name is sometimes
2126 referred to as a *path name component*.

2127 **Note:** Path Name Resolution is defined in detail in Section 4.9 (on page 123).

2128 3.172 File Name Portability

2129 File names should be constructed from the portable file name character set because the use of
2130 other characters can be confusing or ambiguous in certain contexts. (For example, the use of a
2131 colon (':') in a path name could cause ambiguity if that path name were included in a *PATH*
2132 definition.)

2133 3.173 File Offset

2134 The byte position in the file where the next I/O operation begins. Each open file description
2135 associated with a regular file, block special file, or directory has a file offset. A character special
2136 file that does not refer to a terminal device may have a file offset. There is no file offset specified
2137 for a pipe or FIFO.

2138 3.174 File Other Class

2139 The property of a file indicating access permissions for a process related to the user and group
2140 identification of a process. A process is in the file other class of a file if the process is not in the
2141 file owner class or file group class.

2142 3.175 File Owner Class

2143 The property of a file indicating access permissions for a process related to the user
2144 identification of a process. A process is in the file owner class of a file if the effective user ID of
2145 the process matches the user ID of the file.

2146 3.176 File Permission Bits

2147 Information about a file that is used, along with other information, to determine whether a
2148 process has read, write, or execute/search permission to a file. The bits are divided into three
2149 parts: owner, group, and other. Each part is used with the corresponding file class of processes.
2150 These bits are contained in the file mode.

2151 **Note:** File modes are defined in detail in `<sys/stat.h>`.

2152 File Access Permissions are defined in detail in Section 4.3 (on page 121).

2153 3.177 File Serial Number

2154 A per-file system unique identifier for a file.

2155 3.178 File System

2156 A collection of files and certain of their attributes. It provides a name space for file serial
2157 numbers referring to those files.

2158 3.179 File Type

2159 See *File* in Section 3.165 (on page 69).

2160 3.180 Filter

2161 A command whose operation consists of reading data from standard input or a list of input files
2162 and writing data to standard output. Typically, its function is to perform some transformation
2163 on the data stream.

2164 3.181 First Open (of a File)

2165 When a process opens a file that is not currently an open file within any process.

2166 3.182 Flow Control

2167 The mechanism employed by a communications provider that constrains a sending entity to
2168 wait until the receiving entities can safely receive additional data without loss.

2169 3.183 Foreground Job

2170 See *Foreground Process Group* in Section 3.185.

2171 3.184 Foreground Process

2172 A process that is a member of a foreground process group.

2173 3.185 Foreground Process Group (or Foreground Job)

2174 A process group whose member processes have certain privileges, denied to processes in
2175 background process groups, when accessing their controlling terminal. Each session that has
2176 established a connection with a controlling terminal has at most one process group of the session
2177 as the foreground process group of that controlling terminal.

2178 **Note:** The General Terminal Interface is defined in detail in Chapter 11.

2179 3.186 Foreground Process Group ID

2180 The process group ID of the foreground process group.

2181 3.187 Form-Feed Character (<form-feed>)

2182 A character that in the output stream indicates that printing should start on the next page of an
2183 output device. The <form-feed> character is the character designated by '`\f`' in the C language.
2184 If the <form-feed> character is not the first character of an output line, the result is unspecified.
2185 It is unspecified whether this character is the exact sequence transmitted to an output device by
2186 the system to accomplish the movement to the next page.

2187 3.188 Graphic Character

2188 A member of the **graph** character class of the current locale.

2189 **Note:** The **graph** character class is defined in detail in Section 7.3.1 (on page 147).

2190 3.189 Group Database

2191 A system database of implementation-defined format that contains at least the following
2192 information for each group ID:

- 2193 • Group name
- 2194 • Numerical group ID
- 2195 • List of users allowed in the group

2196 The list of users allowed in the group is used by the *newgrp* utility.

2197 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of
2198 IEEE Std. 1003.1-200x.

2199 3.190 Group ID

2200 A non-negative integer, which can be contained in an object of type **gid_t**, that is used to identify
2201 a group of system users. Each system user is a member of at least one group. When the identity
2202 of a group is associated with a process, a group ID value is referred to as a real group ID, an
2203 effective group ID, one of the supplementary group IDs, or a saved set-group-ID.

2204 3.191 Group Name

2205 A string that is used to identify a group; see also Section 3.189. To be portable across conforming
2206 systems, the value is composed of characters from the portable file name character set. The
2207 hyphen should not be used as the first character of a portable group name.

2208 3.192 Hard Limit

2209 A system resource limitation that may be reset to a lesser or greater limit by a privileged process.
2210 A non-privileged process is restricted to only lowering its hard limit.

2211 3.193 Hard Link

2212 The relationship between two directory entries that represent the same file; see also Section 3.132
2213 (on page 63). The result of an execution of the *ln* utility (without the *-s* option) or the *link()*
2214 function. This term is contrasted against symbolic link; see also Section 3.374 (on page 104).

2215 3.194 Home Directory

2216 The directory specified by the *HOME* environment variable.

2217 3.195 Host Byte Order

2218 The arrangement of bytes in any **int** type when using a specific machine architecture.

2219 **Note:** Two common methods of byte ordering are big-endian and little-endian. Big-endian
2220 is a format for storage of binary data in which the most significant byte is placed first,
2221 with the rest in descending order. Little-endian is a format for storage or
2222 transmission of binary data in which the least significant byte is placed first, with the
2223 rest in ascending order.

2224 3.196 Incomplete Line

2225 A sequence of one or more non-`<newline>` characters at the end of the file.

2226 3.197 Inf

2227 A value representing infinity that can be stored in a floating type. Not all systems support the
2228 Inf value.

2229 3.198 Instrumented Application

2230 An application that contains at least one call to the trace point function *posix_trace_event()*. Each
2231 process of an instrumented application has a mapping of trace event names to trace event type
2232 identifiers. This mapping is used by the trace stream that is created for that process.

2233 3.199 Interactive Shell

2234 A processing mode of the shell that is suitable for direct user interaction.

2235 3.200 Internationalization

2236 The provision within a computer program of the capability of making itself adaptable to the
2237 requirements of different native languages, local customs, and coded character sets.

2238 3.201 Interprocess Communication

2239 A functionality enhancement to add a high-performance, deterministic interprocess
2240 communication facility for local communication.

2241 3.202 Invoke

2242 To perform command search and execution actions, except that searching for shell functions and
2243 special built-in utilities is suppressed; see also Section 3.156 (on page 67).

2244 **Note:** Command Search and Execution is defined in detail in the Shell and Utilities volume
2245 of IEEE Std. 1003.1-200x, Section 2.9.1.1, Command Search and Execution.

2246 3.203 Job

2247 A set of processes, comprising a shell pipeline, and any processes descended from it, that are all
2248 in the same process group.

2249 **Note:** See also the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.9.2,
2250 Pipelines.

2251 **3.204 Job Control**

2252 A facility that allows users selectively to stop (suspend) the execution of processes and continue
 2253 (resume) their execution at a later point. The user typically employs this facility via the
 2254 interactive interface jointly supplied by the terminal I/O driver and a command interpreter.

2255 **3.205 Job Control Job ID**

2256 A handle that is used to refer to a job. The job control job ID can be any of the forms shown in the
 2257 following table:

2258 **Table 3-1** Job Control Job ID Formats

2259 2260	Job Control Job ID	Meaning
2261	%%	Current job.
2262	%+	Current job.
2263	%-	Previous job.
2264	% <i>n</i>	Job number <i>n</i> .
2265	% <i>string</i>	Job whose command begins with <i>string</i> .
2266	%? <i>string</i>	Job whose command contains <i>string</i> .

2267 **3.206 Last Close (of a File)**

2268 When a process closes a file, resulting in the file not being an open file within any process.

2269 **3.207 Line**

2270 A sequence of zero or more non-<newline> characters plus a terminating <newline> character.

2271 **3.208 Linger**

2272 Wait for a period of time before terminating a connection, to allow outstanding data to be
 2273 transferred.

2274 **3.209 Link**

2275 See *Directory Entry* in Section 3.132 (on page 63).

2276 **3.210 Link Count**

2277 The number of directory entries that refer to a particular file.

2278 **3.211 Local Customs**

2279 The conventions of a geographical area or territory for such things as date, time, and currency
2280 formats.

2281 **3.212 Local Interprocess Communication (Local IPC)**

2282 The transfer of data between processes in the same system.

2283 **3.213 Locale**

2284 The definition of the subset of a user's environment that depends on language and cultural
2285 conventions.

2286 **Note:** Locales are defined in detail in Chapter 7 (on page 143).

2287 **3.214 Localization**

2288 The process of establishing information within a computer system specific to the operation of
2289 particular native languages, local customs, and coded character sets.

2290 **3.215 Login**

2291 The unspecified activity by which a user gains access to the system. Each login is associated
2292 with exactly one login name.

2293 3.216 Login Name

2294 A user name that is associated with a login.

2295 3.217 Map

2296 To create an association between a page-aligned range of the address space of a process and
2297 some memory object, such that a reference to an address in that range of the address space
2298 results in a reference to the associated memory object. The mapped memory object is not
2299 necessarily memory-resident.

2300 3.218 Marked Message

2301 A STREAMS message on which a certain flag is set. Marking a message gives the application
2302 protocol-specific information. An application can use *ioctl()* to determine whether a given
2303 message is marked.

2304 **Note:** The *ioctl()* function is defined in detail in the System Interfaces volume of
2305 IEEE Std. 1003.1-200x.

2306 3.219 Matched

2307 A state applying to a sequence of zero or more characters when the characters in the sequence
2308 correspond to a sequence of characters defined by a BRE or ERE pattern.

2309 **Note:** Regular Expressions are defined in detail in Chapter 9 (on page 195).

2310 3.220 Memory Mapped Files and Shared Memory Objects

2311 A performance improvement facility to allow for programs to access files as part of the address
2312 space and for separate application programs to have portions of their address space commonly
2313 accessible.

2314 3.221 Memory Object

2315 One of:

- 2316 • A file
- 2317 • A shared memory object
- 2318 • A typed memory object

2319 When used in conjunction with *mmap()*, a memory object appears in the address space of the
2320 calling process.

2321 **Note:** The *mmap()* function is defined in detail in the System Interfaces volume of
2322 IEEE Std. 1003.1-200x.

2323 3.222 Memory-Resident

2324 Managed by the implementation in such a way as to provide an upper bound on memory access
2325 times.

2326 3.223 Message

2327 In the context of programmatic message passing, information that can be transferred between
2328 processes or threads by being added to and removed from a message queue. A message consists
2329 of a fixed-size message buffer.

2330 3.224 Message Catalog

2331 In the context of providing natural language messages to the user, a file or storage area
2332 containing program messages, command prompts, and responses to prompts for a particular
2333 native language, territory, and codeset.

2334 3.225 Message Catalog Descriptor

2335 In the context of providing natural language messages to the user, a per-process unique value
2336 used to identify an open message catalog. A message catalog descriptor may be implemented
2337 using a file descriptor.

2338 3.226 Message Queue

2339 In the context of programmatic message passing, an object to which messages can be added and
2340 removed. Messages may be removed in the order in which they were added or in priority order.

2341 3.227 Mode

2342 A collection of attributes that specifies a file's type and its access permissions.

2343 **Note:** File Access Permissions are defined in detail in Section 4.3 (on page 121).

2344 3.228 Monotonic Clock

2345 A clock whose value cannot be set via `clock_settime()` and which cannot have negative clock
2346 jumps.

2347 3.229 Mount Point

2348 Either the system root directory or a directory for which the `st_dev` field of structure `stat` differs
2349 from that of its parent directory.

2350 **Note:** The `stat` structure is defined in detail in `<sys/stat.h>`.

2351 3.230 Multi-Character Collating Element

2352 A sequence of two or more characters that collate as an entity. For example, in some coded
2353 character sets, an accented character is represented by a non-spacing accent, followed by the
2354 letter. Other examples are the Spanish elements *ch* and *ll*.

2355 3.231 Mutex

2356 A synchronization object used to allow multiple threads to serialize their access to shared data.
2357 The name derives from the capability it provides; namely, mutual-exclusion. The thread that has
2358 locked a mutex becomes its owner and remains the owner until that same thread unlocks the
2359 mutex.

2360 3.232 Name

2361 In the shell command language, a word consisting solely of underscores, digits, and alphabets
2362 from the portable character set. The first character of a name is not a digit.

2363 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 133).

2364 3.233 Named STREAM

2365 A STREAMS-based file descriptor that is attached to a name in the file system name space. All
2366 subsequent operations on the named STREAM act on the STREAM that was associated with the
2367 file descriptor until the name is disassociated from the STREAM.

2368 3.234 NaN (Not a Number)

2369 A value that can be stored in a floating type but that is not a valid floating point number. Not all
2370 systems support the NaN value.

2371 3.235 Native Language

2372 A computer user's spoken or written language, such as American English, British English,
2373 Danish, Dutch, French, German, Italian, Japanese, Norwegian, or Swedish.

2374 3.236 Negative Response

2375 An input string that matches one of the responses acceptable to the *LC_MESSAGES* category
2376 keyword **noexpr**, matching an extended regular expression in the current locale.

2377 **Note:** The *LC_MESSAGES* category is defined in detail in Section 7.3.6 (on page 174).

2378 3.237 Network

2379 A collection of interconnected hosts.

2380 **Note:** The term network in IEEE Std. 1003.1-200x is used to refer to the network of hosts.
2381 The term batch system is used to refer to the network of batch servers.

2382 3.238 Network Address

2383 A network-visible identifier used to designate specific endpoints in a network. Specific
2384 endpoints on host systems have addresses, and host systems may also have addresses.

2385 3.239 Network Byte Order

2386 The way of representing any **int** type such that, when transmitted over a network via a network
2387 endpoint, the **int** type is transmitted as an appropriate number of octets with the most
2388 significant octet first, followed by any other octets in descending order of significance.

2389 **Note:** This order is more commonly known as big-endian ordering.

2390 3.240 Newline Character (<newline>)

2391 A character that in the output stream indicates that printing should start at the beginning of the
2392 next line. The <newline> character is the character designated by ‘\n’ in the C language. It is
2393 unspecified whether this character is the exact sequence transmitted to an output device by the
2394 system to accomplish the movement to the next line.

2395 3.241 Nice Value

2396 A number used as advice to the system to alter process scheduling. Numerically smaller values
2397 give a process additional preference when scheduling a process to run. Numerically larger
2398 values reduce the preference and make a process less likely to run. Typically, a process with a
2399 smaller nice value runs to completion more quickly than an equivalent process with a higher
2400 nice value. The symbol {NZERO} specifies the default nice value of the system.

2401 3.242 Non-Blocking

2402 A property of an open file description that causes it to either perform the requested action or
2403 return an indication that the action could not be immediately performed, in either case returning
2404 without delay (other than normal scheduling delays) from the call.

2405 **Note:** The exact semantics are dependent on the type of file associated with the open file.
2406 For data reads from devices such as ttys and FIFOs, a successful return usually
2407 indicates that data sufficient to satisfy the read was immediately available. Similarly,
2408 for writes, that space to perform (at least part of) the write was available, and for
2409 networking not to await protocol events (for example, acknowledgements) to occur.

2410 3.243 Non-Spacing Characters

2411 A character, such as a character representing a diacritical mark in the ISO/IEC 6937:1994
2412 standard coded character set, which is used in combination with other characters to form
2413 composite graphic symbols.

2414 3.244 NUL

2415 A character with all bits set to zero.

2416 **3.245 Null Byte**

2417 A byte with all bits set to zero.

2418 **3.246 Null Pointer**

2419 The value that is obtained by converting the number 0 into a pointer; for example, **(void *) 0**. The
2420 C language guarantees that this value does not match that of any legitimate pointer, so it is used
2421 by many functions that return pointers to indicate an error.

2422 **3.247 Null String**

2423 See *Empty String* in Section 3.147 (on page 66).

2424 **3.248 Null Wide-Character Code**

2425 A wide-character code with all bits set to zero.

2426 **3.249 Number Sign**

2427 The character '#', also known as *hash sign*.

2428 **3.250 Object File**

2429 A regular file containing the output of a compiler, formatted as input to a linkage editor for
2430 linking with other object files into an executable form. The methods of linking are unspecified
2431 and may involve the dynamic linking of objects at runtime. The internal format of an object file
2432 is unspecified, but a conforming application cannot assume an object file is a text file.

2433 **3.251 Octet**

2434 Unit of data representation that consists of eight contiguous bits.

2435 3.252 Offset Maximum

2436 An attribute of an open file description representing the largest value that can be used as a file
2437 offset.

2438 3.253 Opaque Address

2439 An address such that the entity making use of it requires no details about its contents or format.

2440 3.254 Open File

2441 A file that is currently associated with a file descriptor.

2442 3.255 Open File Description

2443 A record of how a process or group of processes is accessing a file. Each file descriptor refers to
2444 exactly one open file description, but an open file description can be referred to by more than
2445 one file descriptor. A file offset, file status, and file access modes are attributes of an open file
2446 description.

2447 3.256 Operand

2448 An argument to a command that is generally used as an object supplying information to a utility
2449 necessary to complete its processing. Operands generally follow the options in a command line.

2450 **Note:** Utility Argument Syntax is defined in detail in Section 12.1 (on page 227).

2451 3.257 Operator

2452 In the shell command language, either a control operator or a redirection operator.

2453 3.258 Option

2454 An argument to a command that is generally used to specify changes in the utility's default
2455 behavior.

2456 **Note:** Utility Argument Syntax is defined in detail in Section 12.1 (on page 227).

2457 3.259 Option-Argument

2458 A parameter that follows certain options. In some cases an option-argument is included within
2459 the same argument string as the option—in most cases it is the next argument.

2460 **Note:** Utility Argument Syntax is defined in detail in Section 12.1 (on page 227).

2461 3.260 Orientation

2462 A stream has one of three orientations: unoriented, byte-oriented, or wide-oriented.

2463 **Note:** For further information, see the System Interfaces volume of IEEE Std. 1003.1-200x,
2464 Section 2.5.2, Stream Orientation and Encoding Rules.

2465 3.261 Orphaned Process Group

2466 A process group in which the parent of every member is either itself a member of the group or is
2467 not a member of the group's session.

2468 3.262 Page

2469 The granularity of process memory mapping or locking.

2470 Physical memory and memory objects can be mapped into the address space of a process on
2471 page boundaries and in integral multiples of pages. Process address space can be locked into
2472 memory (made memory-resident) on page boundaries and in integral multiples of pages.

2473 3.263 Page Size

2474 The size, in bytes, of the system unit of memory allocation, protection, and mapping. On systems
2475 that have segment rather than page-based memory architectures, the term page means a
2476 segment.

2477 3.264 Parameter

2478 In the shell command language, an entity that stores values. There are three types of parameters:
2479 variables (named parameters), positional parameters, and special parameters. Parameter
2480 expansion is accomplished by introducing a parameter with the '\$' character.

2481 **Note:** See also the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.5,
2482 Parameters and Variables.

2483 In the C language, an object declared as part of a function declaration or definition that acquires
2484 a value on entry to the function, or an identifier following the macro name in a function-like
2485 macro definition.

2486 3.265 Parent Directory

2487 When discussing a given directory, the directory that both contains a directory entry for the
2488 given directory and is represented by the path name dot-dot in the given directory.

2489 When discussing other types of files, a directory containing a directory entry for the file under
2490 discussion.

2491 This concept does not apply to dot and dot-dot.

2492 3.266 Parent Process

2493 The process which created (or inherited) the process under discussion.

2494 3.267 Parent Process ID

2495 An attribute of a new process identifying the parent of the process. The parent process ID of a
2496 process is the process ID of its creator, for the lifetime of the creator. After the creator's lifetime
2497 has ended, the parent process ID is the process ID of an implementation-defined system process.

2498 3.268 Path Name

2499 A character string that is used to identify a file. In the context of IEEE Std. 1003.1-200x, a path
2500 name consists of, at most, {PATH_MAX} bytes, including the terminating null byte. It has an
2501 optional beginning slash, followed by zero or more file names separated by slashes. A path name
2502 may optionally contain one or more trailing slashes. Multiple successive slashes are considered
2503 to be the same as one slash.

2504 **Note:** Path Name Resolution is defined in detail in Section 4.9 (on page 123).

2505 3.269 Path Name Component

2506 See *File Name* in Section 3.171 (on page 70).

2507 3.270 Path Prefix

2508 A path name, with an optional ending slash, that refers to a directory.

2509 3.271 Pattern

2510 A sequence of characters used either with regular expression notation or for path name
2511 expansion, as a means of selecting various character strings or path names, respectively.

2512 **Note:** Regular Expressions are defined in detail in Chapter 9 (on page 195).

2513 See also the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.6.6, Path
2514 Name Expansion.

2515 The syntaxes of the two types of patterns are similar, but not identical; IEEE Std. 1003.1-200x
2516 always indicates the type of pattern being referred to in the immediate context of the use of the
2517 term.

2518 3.272 Period

2519 The character ' . '. The term period is contrasted with dot (see also Section 3.137 (on page 64)),
2520 which is used to describe a specific directory entry.

2521 3.273 Permissions

2522 Attributes of an object that determine the privilege necessary to access or manipulate the object.

2523 **Note:** File Access Permissions are defined in detail in Section 4.3 (on page 121).

2524 3.274 Persistence

2525 A mode for semaphores, shared memory, and message queues requiring that the object and its
2526 state (including data, if any) are preserved after the object is no longer referenced by any process.

2527 Persistence of an object does not imply that the state of the object is maintained across a system
2528 crash or a system reboot.

2529 3.275 Pipe

2530 An object accessed by one of the pair of file descriptors created by the *pipe()* function. Once
2531 created, the file descriptors can be used to manipulate it, and it behaves identically to a FIFO
2532 special file when accessed in this way. It has no name in the file hierarchy.

2533 **Note:** The *pipe()* function is defined in detail in the System Interfaces volume of
2534 IEEE Std. 1003.1-200x.

2535 **3.276 Polling**

2536 A scheduling scheme whereby the local process periodically checks until the prespecified events
2537 (for example, read, write) have occurred.

2538 **3.277 Portable Character Set**

2539 The collection of characters that are required to be present in all locales supported by
2540 conforming systems.

2541 **Note:** The Portable Character Set is defined in detail in Section 6.1 (on page 133).

2542 This term is contrasted against the smaller *portable file name character set*; see also Section 3.278.

2543 **3.278 Portable File Name Character Set**

2544 The set of characters from which portable file names are constructed.

2545 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

2546 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

2547 0 1 2 3 4 5 6 7 8 9 . _ -

2548 The last three characters are the period, underscore, and hyphen characters, respectively.

2549 **3.279 Positional Parameter**

2550 In the shell command language, a parameter denoted by a single digit or one or more digits in
2551 curly braces.

2552 **Note:** For further information, see the Shell and Utilities volume of IEEE Std. 1003.1-200x,
2553 Section 2.5.1, Positional Parameters.

2554 **3.280 Preallocation**

2555 The reservation of resources in a system for a particular use.

2556 Preallocation does not imply that the resources are immediately allocated to that use, but merely
2557 indicates that they are guaranteed to be available in bounded time when needed.

2558 3.281 Preempted Process (or Thread)

2559 A running thread whose execution is suspended due to another thread becoming runnable at a
2560 higher priority.

2561 3.282 Previous Job

2562 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities if the
2563 current job exits. There is at most one previous job; see also Section 3.205 (on page 76).

2564 3.283 Printable Character

2565 One of the characters included in the **print** character classification of the *LC_CTYPE* category in
2566 the current locale.

2567 **Note:** The *LC_CTYPE* category is defined in detail in Section 7.3.1 (on page 147).

2568 3.284 Printable File

2569 A text file consisting only of the characters included in the **print** and **space** character
2570 classifications of the *LC_CTYPE* category and the <backspace> character, all in the current locale.

2571 **Note:** The *LC_CTYPE* category is defined in detail in Section 7.3.1 (on page 147).

2572 3.285 Priority

2573 A non-negative integer associated with processes or threads whose value is constrained to a
2574 range defined by the applicable scheduling policy. Numerically higher values represent higher
2575 priorities.

2576 3.286 Priority Band

2577 The queuing order applied to normal priority STREAMS messages. High priority STREAMS
2578 messages are not grouped by priority bands. The only differentiation made by the STREAMS
2579 mechanism is between zero and non-zero bands, but specific protocol modules may differentiate
2580 between priority bands.

2581 3.287 Priority Inversion

2582 A condition in which a thread that is not voluntarily suspended (waiting for an event or time
2583 delay) is not running while a lower priority thread is running. Such blocking of the higher
2584 priority thread is often caused by contention for a shared resource.

2585 3.288 Priority Scheduling

2586 A performance and determinism improvement facility to allow applications to determine the
2587 order in which threads that are ready to run are granted access to processor resources.

2588 3.289 Priority-Based Scheduling

2589 Scheduling in which the selection of a running thread is determined by the priorities of the
2590 runnable processes or threads.

2591 3.290 Privilege

2592 See *Appropriate Privileges* in Section 3.19 (on page 44).

2593 3.291 Process

2594 An address space with one or more threads executing within that address space, and the
2595 required system resources for those threads.

2596 **Note:** Many of the system resources defined by IEEE Std. 1003.1-200x are shared among all
2597 of the threads within a process. These include the process ID, the parent process ID,
2598 process group ID, session membership, real, effective, and saved-set user ID, real,
2599 effective, and saved-set group ID, supplementary group IDs, current working
2600 directory, root directory, file mode creation mask, and file descriptors.

2601 3.292 Process Group

2602 A collection of processes that permits the signaling of related processes. Each process in the
2603 system is a member of a process group that is identified by a process group ID. A newly created
2604 process joins the process group of its creator.

2605 3.293 Process Group ID

2606 The unique positive integer identifier representing a process group during its lifetime.

2607 **Note:** See also Process Group ID Reuse defined in Section 4.10 (on page 124).

2608 3.294 Process Group Leader

2609 A process whose process ID is the same as its process group ID.

2610 3.295 Process Group Lifetime

2611 A period of time that begins when a process group is created and ends when the last remaining
2612 process in the group leaves the group, due either to the end of the last process' lifetime or to the
2613 last remaining process calling the *setsid()* or *setpgid()* functions.

2614 **Note:** The *setsid()* and *setpgid()* functions are defined in detail in the System Interfaces
2615 volume of IEEE Std. 1003.1-200x.

2616 3.296 Process ID

2617 The unique positive integer identifier representing a process during its lifetime.

2618 **Note:** See also Process ID Reuse defined in Section 4.10 (on page 124).

2619 3.297 Process Lifetime

2620 The period of time that begins when a process is created and ends when its process ID is
2621 returned to the system. After a process is created with a *fork()* function, it is considered active.
2622 At least one thread of control and address space exist until it terminates. It then enters an
2623 inactive state where certain resources may be returned to the system, although some resources,
2624 such as the process ID, are still in use. When another process executes a *wait()*, *waitid()*, or
2625 *waitpid()* function for an inactive process, the remaining resources are returned to the system.
2626 The last resource to be returned to the system is the process ID. At this time, the lifetime of the
2627 process ends.

2628 **Note:** The *fork()*, *wait()*, *waitid()*, and *waitpid()* functions are defined in detail in the System
2629 Interfaces volume of IEEE Std. 1003.1-200x.

2630 3.298 Process Memory Locking

2631 A performance improvement facility to bind application programs into the high-performance
2632 random access memory of a computer system. This avoids potential latencies introduced by the
2633 operating system in storing parts of a program that were not recently referenced on secondary
2634 memory devices.

2635 3.299 Process Termination

2636 There are two kinds of process termination:

- 2637 1. Normal termination occurs by a return from *main()* or when requested with the *exit()* or
2638 *_exit()* functions.
- 2639 2. Abnormal termination occurs when requested by the *abort()* function or when some
2640 signals are received.

2641 **Note:** The *_exit()*, *abort()*, and *exit()* functions are defined in detail in the System Interfaces
2642 volume of IEEE Std. 1003.1-200x.

2643 3.300 Process-To-Process Communication

2644 The transfer of data between processes.

2645 3.301 Process Virtual Time

2646 The measurement of time in units elapsed by the system clock while a process is executing. |

2647 3.302 Program

2648 A prepared sequence of instructions to the system to accomplish a defined task. The term |
2649 program in IEEE Std. 1003.1-200x encompasses applications written in the Shell Command |
2650 Language, complex utility input languages (for example, *awk*, *lex*, *sed*, and so on), and high-level |
2651 languages.

2652 3.303 Protocol

2653 A set of semantic and syntactic rules for exchanging information.

2654 3.304 Pseudo-Terminal

2655 A pseudo-terminal provides the process with an interface that is identical to the terminal
2656 subsystem. A pseudo-terminal is composed of two devices: the *master device* and a *slave device*.
2657 The slave device provides processes with an interface that is identical to the terminal interface,
2658 although there need not be hardware behind that interface. Anything written on the master
2659 device is presented to the slave as an input and anything written on the slave device is presented
2660 as an input on the master side.

2661 3.305 Radix Character

2662 The character that separates the integer part of a number from the fractional part.

2663 3.306 Read-Only File System

2664 A file system that has implementation-defined characteristics restricting modifications.

2665 **Note:** File Times Update is described in detail in Section 4.6 (on page 122).

2666 3.307 Read-Write Lock

2667 Multiple readers, single writer (read-write) locks allow many threads to have simultaneous
2668 read-only access to data while allowing only one thread to have write access at any given time.
2669 They are typically used to protect data that is read-only more frequently than it is changed.

2670 Read-write locks can be used to synchronize threads in the current process and other processes if
2671 they are allocated in memory that is writable and shared among the cooperating processes and
2672 have been initialized for this behavior.

2673 3.308 Real Group ID

2674 The attribute of a process that, at the time of process creation, identifies the group of the user
2675 who created the process; see also Section 3.190 (on page 73).

2676 3.309 Real Time

2677 Time measured as total units elapsed by the system clock without regard to which thread is
2678 executing.

2679 3.310 Realtime Signal Extension

2680 A determinism improvement facility to enable asynchronous signal notifications to an
2681 application to be queued without impacting compatibility with the existing signal functions.

2682 3.311 Real User ID

2683 The attribute of a process that, at the time of process creation, identifies the user who created the
2684 process; see also Section 3.427 (on page 115).

2685 3.312 Record

2686 A collection of related data units or words which is treated as a unit.

2687 3.313 Redirection

2688 In the shell command language, a method of associating files with the input or output of
2689 commands.

2690 **Note:** For further information, see the Shell and Utilities volume of IEEE Std. 1003.1-200x,
2691 Section 2.7, Redirection.

2692 3.314 Redirection Operator

2693 In the shell command language, a token that performs a redirection function. It is one of the
2694 following symbols:

2695 < > >| << >> <& >& <<- <>

2696 3.315 Reentrant Function

2697 A function whose effect, when called by two or more threads, is guaranteed to be as if the
2698 threads each executed the function one after another in an undefined order, even if the actual
2699 execution is interleaved.

2700 3.316 Referenced Shared Memory Object

2701 A shared memory object that is open or has one or more mappings defined on it.

2702 3.317 Refresh

2703 To ensure that the information on the user's terminal screen is up-to-date.

2704 3.318 Regular Expression

2705 A pattern that selects specific strings from a set of character strings.

2706 **Note:** Regular Expressions are described in detail in Chapter 9 (on page 195).

2707 3.319 Region

2708 In the context of the address space of a process, a sequence of addresses.

2709 In the context of a file, a sequence of offsets.

2710 3.320 Regular File

2711 A file that is a randomly accessible sequence of bytes, with no further structure imposed by the
2712 system.

2713 3.321 Relative Path Name

2714 A path name not beginning with a slash.

2715 **Note:** Path Name Resolution is defined in detail in Section 4.9 (on page 123).

2716 3.322 Relocatable File

2717 A file holding code or data suitable for linking with other object files to create an executable or a
2718 shared object file.

2719 3.323 Relocation

2720 The process of connecting symbolic references with symbolic definitions. For example, when a
2721 program calls a function, the associated call instruction transfers control to the proper
2722 destination address at execution.

2723 3.324 Requested Batch Service

2724 A service that is either rejected or performed prior to a response from the service to the
2725 requester.

2726 3.325 (Time) Resolution

2727 The minimum time interval that a clock can measure or whose passage a timer can detect.

2728 3.326 Root Directory

2729 A directory, associated with a process, that is used in path name resolution for path names that
2730 begin with a slash.

2731 3.327 Runnable Process (or Thread)

2732 A thread that is capable of being a running thread, but for which no processor is available.

2733 3.328 Running Process (or Thread)

2734 A thread currently executing on a processor. On multi-processor systems there may be more
2735 than one such thread in a system at a time.

2736 3.329 Saved Resource Limits

2737 An attribute of a process that provides some flexibility in the handling of unrepresentable
2738 resource limits, as described in the *exec* family of functions and *setrlimit()*.

2739 **Note:** The *exec* and *setrlimit()* functions are defined in detail in the System Interfaces
2740 volume of IEEE Std. 1003.1-200x.

2741 3.330 Saved Set-Group-ID

2742 An attribute of a process that allows some flexibility in the assignment of the effective group ID
2743 attribute, as described in the *exec* family of functions and *setgid()*.

2744 **Note:** The *exec* and *setgid()* functions are defined in detail in the System Interfaces volume
2745 of IEEE Std. 1003.1-200x.

2746 3.331 Saved Set-User-ID

2747 An attribute of a process that allows some flexibility in the assignment of the effective user ID
2748 attribute, as described in the *exec* family of functions and *setuid()*.

2749 **Note:** The *exec* and *setuid()* functions are defined in detail in the System Interfaces volume
2750 of IEEE Std. 1003.1-200x.

2751 3.332 Scheduling

2752 The application of a policy to select a runnable process or thread to become a running process or
2753 thread, or to alter one or more of the thread lists.

2754 3.333 Scheduling Allocation Domain

2755 The set of processors on which an individual thread can be scheduled at any given time.

2756 3.334 Scheduling Contention Scope

2757 A property of a thread that defines the set of threads against which that thread competes for
2758 resources.

2759 For example, in a scheduling decision, threads sharing scheduling contention scope compete for
2760 processor resources. In IEEE Std. 1003.1-200x, a thread has scheduling contention scope of either
2761 PTHREAD_SCOPE_SYSTEM or PTHREAD_SCOPE_PROCESS.

2762 3.335 Scheduling Policy

2763 A set of rules that is used to determine the order of execution of processes or threads to achieve
2764 some goal.

2765 **Note:** Scheduling Policy is defined in detail in Section 4.11 (on page 125).

2766 3.336 Screen

2767 A rectangular region of columns and lines on a terminal display. A screen may be a portion of a
2768 physical display device or may occupy the entire physical area of the display device.

2769 3.337 Scroll

2770 To move the representation of data vertically or horizontally relative to the terminal screen.
2771 There are two types of scrolling:

- 2772 1. The cursor moves with the data.
- 2773 2. The cursor remains stationary while the data moves.

2774 3.338 Semaphore

2775 A minimum synchronization primitive to serve as a basis for more complex synchronization
2776 mechanisms to be defined by the application program.

2777 **Note:** Semaphores are defined in detail in Section 4.13 (on page 126).

2778 3.339 Session

2779 A collection of process groups established for job control purposes. Each process group is a
2780 member of a session. A process is considered to be a member of the session of which its process
2781 group is a member. A newly created process joins the session of its creator. A process can alter
2782 its session membership; see *setsid()*. There can be multiple process groups in the same session.

2783 **Note:** The *setsid()* function is defined in detail in the System Interfaces volume of
2784 IEEE Std. 1003.1-200x.

2785 3.340 Session Leader

2786 A process that has created a session.

2787 **Note:** For further information, see the *setsid()* function defined in the System Interfaces
2788 volume of IEEE Std. 1003.1-200x.

2789 3.341 Session Lifetime

2790 The period between when a session is created and the end of the lifetime of all the process
2791 groups that remain as members of the session.

2792 3.342 Shared Memory Object

2793 An object that represents memory that can be mapped concurrently into the address space of
2794 more than one process.

2795 3.343 Shell

2796 A program that interprets sequences of text input as commands. It may operate on an input
2797 stream or it may interactively prompt and read commands from a terminal.

2798 3.344 Shell, the

2799 The Shell Command Language Interpreter; a specific instance of a shell.

2800 **Note:** For further information, see the *sh* utility defined in the Shell and Utilities volume of
2801 IEEE Std. 1003.1-200x.

2802 3.345 Shell Script

2803 A file containing shell commands. If the file is made executable, it can be executed by specifying
2804 its name as a simple command. Execution of a shell script causes a shell to execute the
2805 commands within the script. Alternatively, a shell can be requested to execute the commands in
2806 a shell script by specifying the name of the shell script as the operand to the *sh* utility.

2807 **Note:** Simple Commands are defined in detail in the Shell and Utilities volume of
2808 IEEE Std. 1003.1-200x, Section 2.9.1, Simple Commands.

2809 The *sh* utility is defined in detail in the Shell and Utilities volume of
2810 IEEE Std. 1003.1-200x.

2811 3.346 Signal

2812 A mechanism by which a process or thread may be notified of, or affected by, an event occurring
2813 in the system. Examples of such events include hardware exceptions and specific actions by
2814 processes. The term signal is also used to refer to the event itself.

2815 3.347 Signal Stack

2816 Memory established for a thread, in which signal handlers catching signals sent to that thread
2817 are executed.

2818 3.348 Single-Quote

2819 The character ' ' ', also known as *apostrophe*.

2820 3.349 Slash

2821 The character ' / ', also known as *solidus*.

2822 3.350 Socket

2823 A file of a particular type that is used as a communications endpoint for process-to-process
2824 communication as described in the System Interfaces volume of IEEE Std. 1003.1-200x.

2825 3.351 Socket Address

2826 An address associated with a socket or remote endpoint, including an address family identifier
2827 and addressing information specific to that address family. The address may include multiple
2828 parts, such as a network address associated with a host system and an identifier for a specific
2829 endpoint.

2830 3.352 Soft Limit

2831 A resource limitation established for each process that the process may set to any value less than
2832 or equal to the hard limit.

2833 3.353 Source Code

2834 When dealing with the Shell Command Language, input to the command language interpreter.
2835 The term shell script is synonymous with this meaning.

2836 When dealing with an ISO/IEC-conforming programming language, source code is input to a
2837 compiler conforming to that ISO/IEC standard.

2838 Source code also refers to the input statements prepared for the following standard utilities:
2839 *awk, bc, ed, lex, localedef, make, sed, and yacc*.

2840 Source code can also refer to a collection of sources meeting any or all of these meanings.

2841 **Note:** The *awk, bc, ed, lex, localedef, make, sed, and yacc* utilities are defined in detail in the
2842 Shell and Utilities volume of IEEE Std. 1003.1-200x.

2843 3.354 Space Character (<space>)

2844 The character defined in the portable character set as <space>. The <space> character is a
2845 member of the **space** character class of the current locale, but represents the single character, and
2846 not all of the possible members of the class; see also Section 3.433 (on page 116).

2847 3.355 Spawn

2848 A process creation primitive useful for systems that have difficulty with *fork()* and as an efficient
2849 replacement for *fork()/exec*.

2850 3.356 Special Built-In

2851 See *Built-In Utility* in Section 3.85 (on page 55).

2852 3.357 Special Parameter

2853 In the shell command language, a parameter named by a single character from the following list:

2854 * @ # ? ! - \$ 0

2855 **Note:** For further information, see the Shell and Utilities volume of IEEE Std. 1003.1-200x,
2856 Section 2.5.2, Special Parameters.

2857 3.358 Spin Lock

2858 A synchronization object used to allow multiple threads to serialize their access to shared data.

2859 3.359 Sporadic Server

2860 A scheduling policy for threads and processes that reserves a certain amount of execution
2861 capacity for processing aperiodic events at a given priority level.

2862 3.360 Standard Error

2863 An output stream usually intended to be used for diagnostic messages.

2864 3.361 Standard Input

2865 An input stream usually intended to be used for primary data input.

2866 3.362 Standard Output

2867 An output stream usually intended to be used for primary data output.

2868 3.363 Standard Utilities

2869 The utilities described in the Shell and Utilities volume of IEEE Std. 1003.1-200x.

2870 3.364 Stream

2871 Appearing in lowercase, a stream is a file access object that allows access to an ordered sequence
2872 of characters, as described by the ISO C standard. Such objects can be created by the *fdopen()*,
2873 *fopen()*, or *popen()* functions, and are associated with a file descriptor. A stream provides the
2874 additional services of user-selectable buffering and formatted input and output; see also Section
2875 3.365.

2876 **Note:** For further information, see the System Interfaces volume of IEEE Std. 1003.1-200x,
2877 Section 2.5, Standard I/O Streams.

2878 The *fdopen()*, *fopen()*, or *popen()* functions are defined in detail in the System
2879 Interfaces volume of IEEE Std. 1003.1-200x.

2880 3.365 STREAM

2881 Appearing in uppercase, STREAM refers to a full duplex connection between a process and an
2882 open device or pseudo-device. It optionally includes one or more intermediate processing
2883 modules that are interposed between the process end of the STREAM and the device driver (or
2884 pseudo-device driver) end of the STREAM; see also Section 3.364.

2885 **Note:** For further information, see the System Interfaces volume of IEEE Std. 1003.1-200x,
2886 Section 2.6, STREAMS.

2887 3.366 STREAM End

2888 The STREAM end is the driver end of the STREAM and is also known as the downstream end of
2889 the STREAM.

2890 3.367 STREAM Head

2891 The STREAM head is the beginning of the STREAM and is at the boundary between the system
2892 and the application process. This is also known as the upstream end of the STREAM.

2893 3.368 STREAMS Multiplexor

2894 A driver with multiple STREAMS connected to it. Multiplexing with STREAMS connected above
2895 is referred to as N-to-1, or *upper multiplexing*. Multiplexing with STREAMS connected below is
2896 referred to as 1-to-N or *lower multiplexing*.

2897 3.369 String

2898 A contiguous sequence of bytes terminated by and including the first null byte.

2899 3.370 Subshell

2900 A shell execution environment, distinguished from the main or current shell execution
2901 environment.

2902 **Note:** For further information, see the Shell and Utilities volume of IEEE Std. 1003.1-200x,
2903 Section 2.13, Shell Execution Environment.

2904 3.371 Successfully Transferred

2905 For a write operation to a regular file, when the system ensures that all data written is readable
2906 on any subsequent open of the file (even one that follows a system or power failure) in the
2907 absence of a failure of the physical storage medium.

2908 For a read operation, when an image of the data on the physical storage medium is available to
2909 the requesting process.

2910 3.372 Supplementary Group ID

2911 An attribute of a process used in determining file access permissions. A process has up to
2912 {NGROUPS_MAX} supplementary group IDs in addition to the effective group ID. The
2913 supplementary group IDs of a process are set to the supplementary group IDs of the parent
2914 process when the process is created.

2915 3.373 Suspended Job

2916 A job that has received a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal that caused the
2917 process group to stop. A suspended job is a background job, but a background job is not
2918 necessarily a suspended job.

2919 3.374 Symbolic Link

2920 A type of file with the property that when the file is encountered during path name resolution, a
2921 string stored by the file is used to modify the path name resolution. The stored string has a
2922 length of {SYMLINK_MAX} bytes or fewer.

2923 **Note:** Path Name Resolution is defined in detail in Section 4.9 (on page 123).

2924 3.375 Synchronized Input and Output

2925 A determinism and robustness improvement mechanism to enhance the data input and output
2926 mechanisms, so that an application can ensure that the data being manipulated is physically
2927 present on secondary mass storage devices.

2928 3.376 Synchronized I/O Completion

2929 The state of an I/O operation that has either been successfully transferred or diagnosed as
2930 unsuccessful.

2931 3.377 Synchronized I/O Data Integrity Completion

2932 For read, when the operation has been completed or diagnosed if unsuccessful. The read is
2933 complete only when an image of the data has been successfully transferred to the requesting
2934 process. If there were any pending write requests affecting the data to be read at the time that
2935 the synchronized read operation was requested, these write requests are successfully transferred
2936 prior to reading the data.

2937 For write, when the operation has been completed or diagnosed if unsuccessful. The write is
2938 complete only when the data specified in the write request is successfully transferred and all file
2939 system information required to retrieve the data is successfully transferred.

2940 File attributes that are not necessary for data retrieval (access time, modification time, status
2941 change time) need not be successfully transferred prior to returning to the calling process.

2942 3.378 Synchronized I/O File Integrity Completion

2943 Identical to a synchronized I/O data integrity completion with the addition that all file attributes
2944 relative to the I/O operation (including access time, modification time, status change time) are
2945 successfully transferred prior to returning to the calling process.

2946 3.379 Synchronized I/O Operation

2947 An I/O operation performed on a file that provides the application assurance of the integrity of
2948 its data and files.

2949 3.380 Synchronous I/O Operation

2950 An I/O operation that causes the thread requesting the I/O to be blocked from further use of the
2951 processor until that I/O operation completes.

2952 **Note:** A synchronous I/O operation does not imply synchronized I/O data integrity
2953 completion or synchronized I/O file integrity completion.

2954 3.381 Synchronously-Generated Signal

2955 A signal that is attributable to a specific thread.

2956 For example, a thread executing an illegal instruction or touching invalid memory causes a
2957 synchronously-generated signal. Being synchronous is a property of how the signal was
2958 generated and not a property of the signal number.

2959 3.382 System

2960 An implementation of IEEE Std. 1003.1-200x.

2961 3.383 System Crash

2962 An interval initiated by an unspecified circumstance that causes all processes (possibly other
2963 than special system processes) to be terminated in an undefined manner, after which any
2964 changes to the state and contents of files created or written to by an application prior to the
2965 interval are undefined, except as required elsewhere in IEEE Std. 1003.1-200x.

2966 3.384 System Console

2967 An optional file that receives messages sent by *fmtmsg()* when the MM_CONSOLE flag is set. |

2968 **Note:** The *fmtmsg()* function is defined in detail in the System Interfaces volume of |
2969 IEEE Std. 1003.1-200x.

2970 3.385 System Databases

2971 An implementation provides two system databases.

2972 The *group database* contains the following information for each group:

- 2973 1. Group name
- 2974 2. Numerical group ID
- 2975 3. List of all users allowed in the group

2976 The *user database* contains the following information for each user:

- 2977 1. User name
- 2978 2. Numerical user ID
- 2979 3. Numerical group ID
- 2980 4. Initial working directory
- 2981 5. Initial user program

2982 If the initial user program field is null, the system default is used. If the initial working directory
2983 field is null, the interpretation of that field is implementation-defined. These databases may |
2984 contain other fields that are unspecified by IEEE Std. 1003.1-200x.

2985 3.386 System Documentation

2986 All documentation provided with an implementation except for the conformance document or |
2987 Conformance Statement Questionnaire (CSQ). Electronically distributed documents for an |
2988 implementation are considered part of the system documentation.

2989 3.387 System Process

2990 An implementation-defined object, other than a process executing an application, that has a |
2991 process ID.

2992 3.388 System Reboot

2993 An implementation-defined sequence of events that may result in the loss of transitory data; that
2994 is, data that is not saved in permanent storage. For example, message queues, shared memory,
2995 semaphores, and processes.

2996 3.389 System Trace Event

2997 A trace event that is generated by the implementation, in response either to a system-initiated
2998 action or to an application-requested action, except for a call to *posix_trace_event()*. When
2999 supported by the implementation, a system-initiated action generates a process-independent
3000 system trace event and an application-requested action generates a process-dependent system
3001 trace event. For a system trace event not defined by IEEE Std. 1003.1-200x, the associated trace
3002 event type identifier is derived from the implementation-defined name for this trace event, and
3003 the associated data is of implementation-defined content and length.

3004 3.390 System-Wide

3005 Pertaining to events occurring in all processes existing in an implementation at a given point in
3006 time.

3007 3.391 Tab Character (<tab>)

3008 A character that in the output stream indicates that printing or displaying should start at the
3009 next horizontal tabulation position on the current line. The <tab> character is the character
3010 designated by '\t' in the C language. If the current position is at or past the last defined
3011 horizontal tabulation position, the behavior is unspecified. It is unspecified whether this
3012 character is the exact sequence transmitted to an output device by the system to accomplish the
3013 tabulation.

3014 3.392 Terminal (or Terminal Device)

3015 A character special file that obeys the specifications of the general terminal interface.

3016 **Note:** The General Terminal Interface is defined in detail in Chapter 11 (on page 213).

3017 3.393 Text Column

3018 A roughly rectangular block of characters capable of being laid out side-by-side next to other
3019 text columns on an output page or terminal screen. The widths of text columns are measured in
3020 column positions.

3021 3.394 Text File

3022 A file that contains characters organized into one or more lines. The lines do not contain NUL
3023 characters and none can exceed {LINE_MAX} bytes in length, including the <newline> character.
3024 Although IEEE Std. 1003.1-200x does not distinguish between text files and binary files (see the
3025 ISO C standard), many utilities only produce predictable or meaningful output when operating
3026 on text files. The standard utilities that have such restrictions always specify *text files* in their
3027 STDIN or INPUT FILES sections.

3028 3.395 Thread

3029 A single flow of control within a process. Each thread has its own thread ID, scheduling priority
3030 and policy, *errno* value, thread-specific key/value bindings, and the required system resources to
3031 support a flow of control. Anything whose address may be determined by a thread, including
3032 but not limited to static variables, storage obtained via *malloc()*, directly addressable storage
3033 obtained through implementation-defined functions, and automatic variables, are accessible to
3034 all threads in the same process.

3035 **Note:** The *malloc()* function is defined in detail in the System Interfaces volume of
3036 IEEE Std. 1003.1-200x.

3037 3.396 Thread ID

3038 Each thread in a process is uniquely identified during its lifetime by a value of type **pthread_t**
3039 called a thread ID.

3040 3.397 Thread List

3041 An ordered set of runnable threads that all have the same ordinal value for their priority.

3042 The ordering of threads on the list is determined by a scheduling policy or policies. The set of
3043 thread lists includes all runnable threads in the system.

3044 3.398 Thread-Safe

3045 A function that may be safely invoked concurrently by multiple threads. Each function defined
3046 in the System Interfaces volume of IEEE Std. 1003.1-200x is thread-safe unless explicitly stated
3047 otherwise. Examples are any “pure” function, a function which holds a mutex locked while it is
3048 accessing static storage, or objects shared among threads.

3049 3.399 Thread-Specific Data Key

3050 A process global handle of type `pthread_key_t` which is used for naming thread-specific data.

3051 Although the same key value may be used by different threads, the values bound to the key by
3052 `pthread_setspecific()` and accessed by `pthread_getspecific()` are maintained on a per-thread basis
3053 and persist for the life of the calling thread.

3054 **Note:** The `pthread_getspecific()` and `pthread_setspecific()` functions are defined in detail in the
3055 System Interfaces volume of IEEE Std. 1003.1-200x.

3056 3.400 Tilde

3057 The character ‘~’.

3058 3.401 Timeouts

3059 A method of limiting the length of time an interface will block; see also Section 3.77 (on page 54).

3060 3.402 Timer

3061 A mechanism that can notify a thread when the time as measured by a particular clock has
3062 reached or passed a specified value, or when a specified amount of time has passed.

3063 3.403 Timer Overrun

3064 A condition that occurs each time a timer, for which there is already an expiration signal queued
3065 to the process, expires.

3066 **3.404 Token**

3067 In the shell command language, a sequence of characters that the shell considers as a single unit
3068 when reading input. A token is either an operator or a word.

3069 **Note:** The rules for reading input are defined in detail in the Shell and Utilities volume of
3070 IEEE Std. 1003.1-200x, Section 2.3, Token Recognition.

3071 **3.405 Trace Analyzer Process**

3072 A process that extracts trace events from a trace stream to retrieve information about the
3073 behavior of an application. A trace controller process may also be a trace analyzer process. Trace
3074 analysis can be done concurrently with the traced process or can be done off-line, in the same or
3075 in a different platform.

3076 **3.406 Trace Controller Process**

3077 A process that creates a trace stream for tracing a process. Only the trace controller process has
3078 control of the trace stream it has created. The control of the operation of a trace stream is done
3079 using its corresponding trace stream identifier. The trace controller process is able to:

- 3080 • Initialize the attributes of a trace stream
- 3081 • Create the trace stream
- 3082 • Start and stop tracing
- 3083 • Know the mapping of the traced process
- 3084 • If the Trace Event Filter option is supported, filter the type of trace events to be recorded
- 3085 • Shut the trace stream down

3086 A traced process may also be a trace controller process. Only the trace controller process can
3087 control its trace stream(s). A trace stream created by a trace controller process is shut down if its
3088 controller process terminates or executes another file.

3089 **3.407 Trace Event**

3090 A data object that represents an action executed by the system, and that is recorded in a trace
3091 stream. Each trace event is of a particular trace event type, and is associated with a trace event
3092 type identifier. The execution of a trace point generates a trace event if a trace stream has been
3093 created and started for the process that executed the trace point and if the corresponding trace
3094 event type identifier is not ignored by filtering.

3095 A generated trace event should be recorded in a trace stream and optionally also in a trace log if
3096 a trace log was associated with the trace stream.

3097 The only case in which a generated trace event is not recorded in the trace stream is when no
3098 resources are available for it in the trace stream. In this case, the trace event is lost.

3099 The only two cases in which a generated trace event is not recorded in the trace log are when no
3100 resources are available for it in the trace log or when a flush operation does not succeed.

3101 A trace event recorded in an active trace stream may be retrieved by an application having the
3102 appropriate privileges.

3103 A trace event recorded in a trace log may be retrieved by an application having the appropriate
3104 privileges after opening the trace log as a pre-recorded trace stream, with the function
3105 *posix_trace_open()*.

3106 When a trace event is reported it is possible to retrieve the following:

- 3107 • A trace event type identifier
- 3108 • A timestamp
- 3109 • The process ID of the traced process, if the trace event is process-dependent
- 3110 • Any optional trace event data including its length
- 3111 • If the Threads option is supported, the thread ID, if the trace event is process-dependent
- 3112 • The program address at which the trace point was invoked

3113 **3.408 Trace Event Type**

3114 A data object type that defines a class of trace event. A trace event type is identified on the one
3115 hand by a trace event type name, also referenced as a trace event name, and on the other hand by
3116 a trace event type identifier. A trace event name is a human-readable string. A trace event type
3117 identifier is an opaque identifier used by the trace system. There is a one-to-one relationship
3118 between trace event type identifiers and trace event names for a given trace stream and also for a
3119 given traced process. The trace event type identifier is generated automatically from a trace
3120 event name by the trace system either when a trace controller process invokes
3121 *posix_trace_trid_eventid_open()* or when an instrumented application process invokes
3122 *posix_trace_eventid_open()*. Trace event type identifiers are used to filter trace event types, to
3123 allow interpretation of user data, and to identify the kind of trace point that generated a trace
3124 event.

3125 **3.409 Trace Event Type Mapping**

3126 A one-to-one mapping between trace event types and trace event names. One such mapping is
3127 associated with each trace stream. An active trace stream is associated to a traced process, and
3128 also to its children if the Trace Inherit option is supported and also the inheritance policy is set to
3129 `_POSIX_TRACE_INHERIT`. Therefore each traced process has a mapping of the trace event
3130 names to trace event type identifiers that have been defined for that process.

3131 3.410 Trace Filter

3132 A filter that allows the trace controller process to specify those trace event types that are to be
3133 ignored; that is, not generated. The operation of the filter is to filter out (ignore) selected trace
3134 events. By default, no trace events are filtered.

3135 3.411 Trace Generation Version

3136 A data object that is an implementation-defined character string, generated by the trace system
3137 and describing the origin and version of the trace system.

3138 3.412 Trace Log

3139 The flushed image of a trace stream, if the trace stream is created with a trace log. The trace log
3140 is recorded when the *posix_trace_shutdown()* operation is invoked or during tracing, depending
3141 on the tracing strategy which is defined by a log policy. After the trace stream has been shut
3142 down, the trace information can be retrieved from the associated trace log using the same
3143 interface used to retrieve information from an active trace stream.

3144 3.413 Trace Point

3145 An action that may cause a trace event to be generated. This may be an implementation-defined
3146 action such as a context switch, or an application-programmed action such as a call to a specific
3147 operating system service (for example, *fork()*) or a call to *posix_trace_event()*.

3148 3.414 Trace Stream

3149 An opaque object that contains trace events plus internal data needed to interpret those trace
3150 events. The implementation and format of a trace stream are unspecified. A trace stream need
3151 not be and generally is not persistent. A trace stream may be either active or pre-recorded:

- 3152 • An active trace stream is a trace stream that has been created and has not yet been shut
3153 down. It can be of one of the two following classes:
 - 3154 1. An active trace stream without a trace log that was created with the *posix_trace_create()*
3155 function
 - 3156 2. If the Trace Log option is supported, an active trace stream with a trace log that was
3157 created with the *posix_trace_create_withlog()* function
- 3158 • A pre-recorded trace stream is a trace stream that was opened from a trace log object using
3159 the *posix_trace_open()* function.

3160 An active trace stream can loop. This behavior means that when the resources allocated by the
3161 trace system for the trace stream are exhausted, the trace system reuses the resources associated
3162 with the oldest recorded trace events to record new trace events.

3163 If the Trace Log option is supported, an active trace stream with a trace log can be flushed. This
3164 operation causes the trace system to write trace events from the trace stream to the associated
3165 trace log, following the defined policies or using an explicit function call. After this operation,
3166 the trace system may reuse the resources associated with the flushed trace events.

3167 An active trace stream with or without a trace log can be cleared. This operation causes all the
3168 resources associated with this trace stream to be reinitialized. The trace stream behaves as if it
3169 was returning from its creation, except that the mapping of trace event type identifiers to trace
3170 event names is not cleared. If a trace log was associated with this trace stream, the trace log is
3171 also reinitialized.

3172 **3.415 Trace Stream Identifier**

3173 A handle to manage tracing operations in a trace stream.

3174 **3.416 Trace System**

3175 A system that allows both system and user trace events to be generated into a trace stream.
3176 These trace events can be retrieved later.

3177 **3.417 Traced Process**

3178 A process for which at least one trace stream has been created. A traced process is also called a
3179 target process. If the Trace Inherit option is supported and the trace stream's inheritance
3180 attribute is `_POSIX_TRACE_INHERIT`, the initial targeted traced process is traced together with
3181 all of its future children. The *posix_pid* member of each trace event in a trace stream is the
3182 process ID of the traced process.

3183 **3.418 Tracing Status of a Trace Stream**

3184 A status that describes the state of an active trace stream. The tracing status of a trace stream can
3185 be retrieved from the trace stream attributes. An active trace stream can be in one of two states:
3186 running or suspended.

3187 **3.419 Typed Memory Name Space**

3188 A system-wide name space that contains the names of the typed memory objects present in the
3189 system. It is configurable for a given implementation.

3190 3.420 Typed Memory Object

3191 A combination of a typed memory pool and a typed memory port. The entire contents of the
3192 pool are accessible from the port. The typed memory object is identified through a name that
3193 belongs to the typed memory name space.

3194 3.421 Typed Memory Pool

3195 An extent of memory with the same operational characteristics. Typed memory pools may be
3196 contained within each other.

3197 3.422 Typed Memory Port

3198 A hardware access path to one or more typed memory pools.

3199 3.423 Unbind

3200 Remove the association between a network address and an endpoint.

3201 3.424 Unit Data

3202 See *Datagram* in Section 3.126 (on page 62).

3203 3.425 Upshifting

3204 The conversion of a lowercase character that has a single-character uppercase representation
3205 into this uppercase representation.

3206 3.426 User Database

3207 A system database of implementation-defined format that contains at least the following
3208 information for each user ID:

- 3209 • User name
- 3210 • Numerical user ID
- 3211 • Initial numerical group ID
- 3212 • Initial working directory
- 3213 • Initial user program

3214 The initial numerical group ID is used by the *newgrp* utility. Any other circumstances under
3215 which the initial values are operative are implementation-defined.

3216 If the initial user program field is null, an implementation-defined program is used.

3217 If the initial working directory field is null, the interpretation of that field is implementation-
3218 defined.

3219 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of
3220 IEEE Std. 1003.1-200x.

3221 **3.427 User ID**

3222 A non-negative integer that is used to identify a system user. When the identity of a user is
3223 associated with a process, a user ID value is referred to as a real user ID, an effective user ID, or a
3224 saved set-user-ID.

3225 **3.428 User Name**

3226 A string that is used to identify a user; see also Section 3.426 (on page 114). To be portable across
3227 systems conforming to IEEE Std. 1003.1-200x, the value is composed of characters from the
3228 portable file name character set. The hyphen should not be used as the first character of a
3229 portable user name.

3230 **3.429 User Trace Event**

3231 A trace event that is generated explicitly by the application as a result of a call to
3232 *posix_trace_event()*.

3233 **3.430 Utility**

3234 A program, excluding special built-in utilities provided as part of the Shell Command Language,
3235 that can be called by name from a shell to perform a specific task, or related set of tasks.

3236 **Note:** For further information on special built-in utilities, see the Shell and Utilities volume
3237 of IEEE Std. 1003.1-200x, Section 2.15, Special Built-In Utilities.

3238 **3.431 Variable**

3239 In the shell command language, a named parameter.

3240 **Note:** For further information, see the Shell and Utilities volume of IEEE Std. 1003.1-200x,
3241 Section 2.5, Parameters and Variables.

3242 3.432 Vertical-Tab Character (<vertical-tab>)

3243 A character that in the output stream indicates that printing should start at the next vertical
3244 tabulation position. The <vertical-tab> character is the character designated by '\v' in the C
3245 language. If the current position is at or past the last defined vertical tabulation position, the
3246 behavior is unspecified. It is unspecified whether this character is the exact sequence transmitted
3247 to an output device by the system to accomplish the tabulation.

3248 3.433 White Space

3249 A sequence of one or more characters that belong to the **space** character class as defined via the
3250 *LC_CTYPE* category in the current locale.

3251 In the POSIX locale, white space consists of one or more <blank> characters (<space> and <tab>
3252 characters), <newline> characters, <carriage-return> characters, <form-feed> characters, and
3253 <vertical-tab> characters.

3254 3.434 Wide-Character Code (C Language)

3255 An integer value corresponding to a single graphic symbol or control code.

3256 **Note:** C Language Wide-Character Codes are defined in detail in Section 6.3 (on page 137).

3257 3.435 Wide-Character Input/Output Functions

3258 The functions that perform wide-oriented input from streams or wide-oriented output to
3259 streams: *fgetwc()*, *fputwc()*, *fputws()*, *fwprintf()*, *fwscanf()*, *getwc()*, *getwchar()*, *getws()*, *putwc()*,
3260 *putwchar()*, *ungetwc()*, *vfwprintf()*, *vwprintf()*, *wprintf()*, and *wscanf()*.

3261 **Note:** These functions are defined in detail in the System Interfaces volume of
3262 IEEE Std. 1003.1-200x.

3263 3.436 Wide-Character String

3264 A contiguous sequence of wide-character codes terminated by and including the first null wide-
3265 character code.

3266 3.437 Word

3267 In the shell command language, a token other than an operator. In some cases a word is also a
3268 portion of a word token: in the various forms of parameter expansion, such as $\${name-word}$, and
3269 variable assignment, such as $name=word$, the word is the portion of the token depicted by *word*.
3270 The concept of a word is no longer applicable following word expansions—only fields remain.

3271 **Note:** For further information, see the Shell and Utilities volume of IEEE Std. 1003.1-200x,
3272 Section 2.6.2, Parameter Expansion and the Shell and Utilities volume of
3273 IEEE Std. 1003.1-200x, Section 2.6, Word Expansions.

3274 3.438 Working Directory (or Current Working Directory)

3275 A directory, associated with a process, that is used in path name resolution for path names that
3276 do not begin with a slash.

3277 3.439 Worldwide Portability Interface

3278 Functions for handling characters in a codeset-independent manner.

3279 3.440 Write

3280 To output characters to a file, such as standard output or standard error. Unless otherwise
3281 stated, standard output is the default output destination for all uses of the term write; see the
3282 distinction between display and write in Section 3.135 (on page 64).

3283 3.441 XSI

3284 The X/Open System Interface is the core application programming interface for C and *sh*
3285 programming for systems conforming to the Single UNIX Specification. This is a superset of the
3286 mandatory requirements for conformance to IEEE Std. 1003.1-200x.

3287 3.442 XSI-Conformant

3288 A system which allows an application to be built using a set of services that are consistent across
3289 all systems that conform to IEEE Std. 1003.1-200x and that support the XSI extension.

3290 **Note:** See also Chapter 2 (on page 19).

3291 **3.443 Zombie Process**

3292 A process that has terminated and that is deleted when its exit status has been reported to
 3293 another process which is waiting for that process to terminate.

3294 **3.444 ±0**

3295 The algebraic sign provides additional information about any variable that has the value zero
 3296 when the representation allows the sign to be determined.

3297 **CHANGE HISTORY**3298 **Issue 4**

3299 Numerous changes and additions are made for alignment with the ISO C standard and
 3300 the ISO POSIX-1 standard.

3301 **Issue 4, Version 2**

3302 The following terms are added to support the adoption of additional traditional UNIX
 3303 interfaces: *alternate signal stack*, *break value*, *data segment*, *driver*, *hard limit*, *host byte*
 3304 *order*, *named STREAM*, *network byte order*, *network host database*, *network net database*,
 3305 *network protocol database*, *network service database*, *pad*, *parent window*, *priority band*,
 3306 *process virtual time*, *pseudo-terminal*, *real time*, *signal stack*, *socket*, *soft limit*, *STREAM*
 3307 (second definition), *STREAM end*, *STREAM head*, *STREAMS multiplexor*, *symbolic link*,
 3308 *system console*, and *timer*.

3309 **Issue 5**

3310 Numerous terms are added to support adoption of the POSIX Threads Extension and
 3311 the POSIX Realtime Extension.

3312 **Issue 6**

3313 Additional terms are added to cover material from the ISO POSIX-1: 1996 standard and
 3314 the ISO POSIX-2: 1993 standard not previously included.

3315 Various XSI-related terms are added.

3316 The following definitions are added for alignment with IEEE Std. 1003.1d-1999: *Spawn*,
 3317 *Timeouts*, *Execution Time Monitoring*, *Sporadic Server*, *Advisory Information*, *CPU*
 3318 *Time*, *CPU-Time Clock*, *CPU-Time Timer*, and *Execution Time*.

3319 The definition of *Memory Object* is modified to include typed memory objects for
 3320 alignment with IEEE Std. 1003.1j-2000.

3321 Definitions of *Barrier*, *Clock Jump*, *Monotonic Clock*, *Read-Write Lock*, *Spin Lock*,
 3322 *Typed Memory Name Space*, *Typed Memory Object*, *Typed Memory Pool*, and *Typed*
 3323 *Memory Port* are added for alignment with IEEE Std. 1003.1j-2000.

3324 The *Read-Write Lock* definition is moved under the *RWL* option for alignment with
 3325 IEEE Std. 1003.1j-2000.

3326 **Notes to Reviewers**

3327 *This section with side shading will not appear in the final copy. - Ed.*

3328 To be further expanded.

3330

3331 4.1 Concurrent Execution

3332 Functions that suspend the execution of the calling thread shall not cause the execution of other
3333 threads to be indefinitely suspended.

3334 4.2 Extended Security Controls

3335 An implementation may provide implementation-defined extended security controls (see
3336 Section 3.161 (on page 68)). These permit an implementation to provide security mechanisms to
3337 implement different security policies than those described in IEEE Std. 1003.1-200x. These
3338 mechanisms shall not alter or override the defined semantics of any of the interfaces in
3339 IEEE Std. 1003.1-200x.

3340 4.3 File Access Permissions

3341 The standard file access control mechanism uses the file permission bits, as described below.

3342 Implementations may provide *additional* or *alternate* file access control mechanisms, or both. An
3343 additional access control mechanism shall only further restrict the access permissions defined by
3344 the file permission bits. An alternate file access control mechanism shall:

- 3345 • Specify file permission bits for the file owner class, file group class, and file other class of that
3346 file, corresponding to the access permissions.
- 3347 • Be enabled only by explicit user action, on a per-file basis by the file owner or a user with the
3348 appropriate privilege.
- 3349 • Be disabled for a file after the file permission bits are changed for that file with *chmod()*. The
3350 disabling of the alternate mechanism need not disable any additional mechanisms supported
3351 by an implementation.

3352 Whenever a process requests file access permission for read, write, or execute/search, if no
3353 additional mechanism denies access, access is determined as follows:

- 3354 • If a process has the appropriate privilege:
 - 3355 — If read, write, or directory search permission is requested, access is granted.
 - 3356 — If execute permission is requested, access is granted if execute permission is granted to at
3357 least one user by the file permission bits or by an alternate access control mechanism;
3358 otherwise, access is denied.
- 3359 • Otherwise:
 - 3360 — The file permission bits of a file contain read, write, and execute/search permissions for
3361 the file owner class, file group class, and file other class.
 - 3362 — Access is granted if an alternate access control mechanism is not enabled and the
3363 requested access permission bit is set for the class (file owner class, file group class, or file
3364 other class) to which the process belongs, or if an alternate access control mechanism is

3365 enabled and it allows the requested access; otherwise, access is denied.

3366 **4.4 File Hierarchy**

3367 Files in the system are organized in a hierarchical structure in which all of the non-terminal
3368 nodes are directories and all of the terminal nodes are any other type of file. Because multiple
3369 directory entries may refer to the same file, the hierarchy is properly described as a *directed*
3370 *graph*.

3371 **4.5 File Names**

3372 For a file name to be portable across implementations conforming to IEEE Std. 1003.1-200x, it
3373 shall consist only of the Portable File Name Character Set as defined in Section 3.278 (on page
3374 88).

3375 The hyphen character shall not be used as the first character of a portable file name. Uppercase
3376 and lowercase letters retain their unique identities between conforming implementations. In the
3377 case of a portable path name, the slash character may also be used.

3378 **4.6 File Times Update**

3379 Each file has three distinct associated time values: *st_atime*, *st_mtime*, and *st_ctime*. The *st_atime*
3380 field is associated with the times that the file data is accessed; *st_mtime* is associated with the
3381 times that the file data is modified; and *st_ctime* is associated with the times that the file status is
3382 changed. These values are returned in the file characteristics structure, as described in
3383 `<sys/stat.h>`.

3384 Each function or utility in IEEE Std. 1003.1-200x that reads or writes data or changes file status
3385 indicates which of the appropriate time-related fields shall be “marked for update”. If an
3386 implementation of such a function or utility marks for update a time-related field not specified
3387 by IEEE Std. 1003.1-200x, this shall be documented, except that any changes caused by path
3388 name resolution need not be documented. For the other functions or utilities in
3389 IEEE Std. 1003.1-200x (those that are not explicitly required to read or write file data or change
3390 file status, but that in some implementations happen to do so), the effect is unspecified.

3391 An implementation may update fields that are marked for update immediately, or it may update
3392 such fields periodically. At an update point in time, any marked fields are set to the current time
3393 and the update marks are cleared. All fields that are marked for update shall be updated when
3394 the file ceases to be open by any process, or when a *stat()*, *lstat()*, or *lstat()* is performed on the
3395 file. Other times at which updates are done are unspecified. Marks for update, and updates
3396 themselves, are not done for files on read-only file systems; see Section 3.306 (on page 93).

3397 4.7 Measurement of Execution Time

3398 The mechanism used to measure execution time shall be implementation-defined. The
 3399 implementation shall also define to whom the CPU time that is consumed by interrupt handlers
 3400 and system services on behalf of the operating system will be charged. See Section 3.120 (on
 3401 page 62).

3402 4.8 Memory Synchronization

3403 Applications shall ensure that access to any memory location by more than one thread of control
 3404 (threads or processes) is restricted such that no thread of control can read or modify a memory
 3405 location while another thread of control may be modifying it. Such access is restricted using
 3406 functions that synchronize thread execution and also synchronize memory with respect to other
 3407 threads. The following functions synchronize memory with respect to other threads:

3408	<code>fork()</code>	<code>pthread_mutex_trylock()</code>		
3409	<code>pthread_cond_broadcast()</code>	<code>pthread_mutex_unlock()</code>		
3410	<code>pthread_cond_signal()</code>	<code>sem_post()</code>		
3411	<code>pthread_cond_timedwait()</code>	<code>sem_trywait()</code>		
3412	<code>pthread_cond_wait()</code>	<code>sem_wait()</code>		
3413	<code>pthread_create()</code>	<code>wait()</code>		
3414	<code>pthread_join()</code>	<code>waitpid()</code>		
3415	<code>pthread_mutex_lock()</code>			

3416 Notes to Reviewers

3417 *This section with side shading will not appear in the final copy. - Ed.*

3418 We need to check whether there should be any additional functions listed.

3419 Unless explicitly stated otherwise, if one of the above functions returns an error, it is unspecified
 3420 whether the invocation causes memory to be synchronized.

3421 Applications may allow more than one thread of control to read a memory location
 3422 simultaneously.

3423 4.9 Path Name Resolution

3424 Path name resolution is performed for a process to resolve a path name to a particular file in a
 3425 file hierarchy. There may be multiple path names that resolve to the same file.

3426 Each file name in the path name is located in the directory specified by its predecessor (for
 3427 example, in the path name fragment **a/b**, file **b** is located in directory **a**). Path name resolution
 3428 fails if this cannot be accomplished. If the path name begins with a slash, the predecessor of the
 3429 first file name in the path name is taken to be the root directory of the process (such path names
 3430 are referred to as *absolute path names*). If the path name does not begin with a slash, the
 3431 predecessor of the first file name of the path name is taken to be the current working directory of
 3432 the process (such path names are referred to as *relative path names*).

3433 The interpretation of a path name component is dependent on the value of {NAME_MAX} and
 3434 _POSIX_NO_TRUNC associated with the path prefix of that component. If any path name
 3435 component is longer than {NAME_MAX}, the implementation shall consider this an error.

3436 A path name that contains at least one non-slash character and that ends with one or more
 3437 trailing slashes shall be resolved as if a single dot character ('.') were appended to the path

3438 name.

3439 If a symbolic link is encountered during path name resolution, the behavior shall depend on
3440 whether the path name component is at the end of the path name and on the function being
3441 performed. If all of the following are true, then path name resolution is complete:

- 3442 1. This is the last path name component of the path name.
- 3443 2. The path name has no trailing slash.
- 3444 3. The function is required to act on the symbolic link itself, or certain arguments direct that
3445 the function act on the symbolic link itself.

3446 In all other cases, the system shall prefix the remaining path name, if any, with the contents of
3447 the symbolic link. If the combined length exceeds {PATH_MAX}, and the implementation
3448 considers this to be an error, *errno* shall be set to [ENAMETOOLONG] and an error indication
3449 shall be returned. Otherwise, the resolved path name shall be the resolution of the path name
3450 just created. If the resulting path name does not begin with a slash, the predecessor of the first
3451 file name of the path name is taken to be the directory containing the symbolic link.

3452 If the system detects a loop in the path name resolution process, it shall set *errno* to [ELOOP] and
3453 return an error indication. The same may happen if during the resolution process more symbolic
3454 links were followed than the implementation allows. This implementation-defined limit shall
3455 not be smaller than {SYMLOOP_MAX}.

3456 The special file name dot refers to the directory specified by its predecessor. The special file
3457 name dot-dot refers to the parent directory of its predecessor directory. As a special case, in the
3458 root directory, dot-dot may refer to the root directory itself.

3459 A path name consisting of a single slash resolves to the root directory of the process. A null path
3460 name shall not be successfully resolved. A path name that begins with two successive slashes
3461 may be interpreted in an implementation-defined manner, although more than two leading
3462 slashes shall be treated as a single slash.

3463 **4.10 Process ID Reuse**

3464 A process group ID shall not be reused by the system until the process group lifetime ends.

3465 A process ID shall not be reused by the system until the process lifetime ends. In addition, if
3466 there exists a process group whose process group ID is equal to that process ID, the process ID
3467 shall not be reused by the system until the process group lifetime ends. A process that is not a
3468 system process shall not have a process ID of 1.

3469 **4.11 Scheduling Policy**

3470 A scheduling policy affects process or thread ordering:

- 3471 • When a process or thread is a running thread and it becomes a blocked thread
- 3472 • When a process or thread is a running thread and it becomes a preempted thread
- 3473 • When a process or thread is a blocked thread and it becomes a runnable thread
- 3474 • When a running thread calls a function that can change the priority or scheduling policy of a
- 3475 process or thread
- 3476 • In other scheduling policy-defined circumstances

3477 Conforming implementations are required to define the manner in which each of the scheduling
 3478 policies may modify the priorities or otherwise affect the ordering of processes or threads at
 3479 each of the occurrences listed above. Additionally, conforming implementations define in what
 3480 other circumstances and in what manner each scheduling policy may modify the priorities or
 3481 affect the ordering of processes or threads.

3482 **4.12 Seconds Since the Epoch**

3483 A value that approximates the number of seconds that have elapsed since the Epoch. A
 3484 Coordinated Universal Time name (specified in terms of seconds (*tm_sec*), minutes (*tm_min*),
 3485 hours (*tm_hour*), days since January 1 of the year (*tm_yday*), and calendar year minus 1900
 3486 (*tm_year*)) is related to a time represented as seconds since the Epoch, according to the
 3487 expression below.

3488 If the year is <1970 or the value is negative, the relationship is undefined. If the year is ≥1970 and
 3489 the value is non-negative, the value is related to a Coordinated Universal Time name according
 3490 to the C-language expression, where *tm_sec*, *tm_min*, *tm_hour*, *tm_yday*, and *tm_year* are all
 3491 integer types:

$$\begin{aligned}
 &tm_sec + tm_min*60 + tm_hour*3600 + tm_yday*86400 + \\
 & (tm_year-70)*31536000 + ((tm_year-69)/4)*86400 - \\
 & ((tm_year-1/100)*86400 + ((tm_year+299)/400)*86400
 \end{aligned}$$

3495 Whether and how the implementation accounts for leap seconds is unspecified.

3496 **Note:** The last term of the current expression adds in a day for every 4th year starting in
 3497 1973. (January 1st of each year following a leap year starting with the first leap year
 3498 after 1970). The first term above subtracts a day every 100 years starting in 2001. The
 3499 last term above adds a day back in every 400 years starting in 2001.

3500 4.13 Semaphore

3501 A minimum synchronization primitive to serve as a basis for more complex synchronization
3502 mechanisms to be defined by the application program.

3503 For the semaphores associated with the Semaphores option, a semaphore is represented as a
3504 shareable resource that has a non-negative integral value. When the value is zero, there is a
3505 (possibly empty) set of threads awaiting the availability of the semaphore.

3506 For the semaphores associated with the X/Open System Interface Extension (XSI), a semaphore
3507 is a positive integer (0 through 32767). The *semget()* function can be called to create a set or array
3508 of semaphores. A semaphore set can contain one or more semaphores up to an implementation-
3509 defined value.

3510 Semaphore Lock Operation

3511 An operation that is applied to a semaphore. If, prior to the operation, the value of the
3512 semaphore is zero, the semaphore lock operation shall cause the calling thread to be blocked and
3513 added to the set of threads awaiting the semaphore; otherwise, the value is decremented.

3514 Semaphore Unlock Operation

3515 An operation that is applied to a semaphore. If, prior to the operation, there are any threads in
3516 the set of threads awaiting the semaphore, then some thread from that set shall be removed from
3517 the set and becomes unblocked; otherwise, the semaphore value is incremented.

3518 4.14 Thread-Safety

3519 Refer to the System Interfaces volume of IEEE Std. 1003.1-200x, Section 2.9, Threads.

3520 4.15 Utility

3521 A utility program shall be either an executable file, such as might be produced by a compiler or
3522 linker system from computer source code, or a file of shell source code, directly interpreted by
3523 the shell. The program may have been produced by the user, provided by the system
3524 implementor, or acquired from an independent distributor.

3525 The system may implement certain utilities as shell functions (see the Shell and Utilities volume
3526 of IEEE Std. 1003.1-200x, Section 2.9.5, Function Definition Command) or built-in utilities, but
3527 only an application that is aware of the command search order described in the Shell and
3528 Utilities volume of IEEE Std. 1003.1-200x, Section 2.9.1.1, Command Search and Execution or of
3529 performance characteristics can discern differences between the behavior of such a function or
3530 built-in utility and that of an executable file.

3531 **4.16 Variable Assignment**

3532 In the shell command language, a word consisting of the following parts:

3533 *varname=value*

3534 When used in a context where assignment is defined to occur and at no other time, the *value*
3535 (representing a word or field) shall be assigned as the value of the variable denoted by *varname*.

3536 **Note:** For further information, see the Shell and Utilities volume of IEEE Std. 1003.1-200x,
3537 Section 2.9.1, Simple Commands.

3538 The *varname* and *value* parts meet the requirements for a name and a word, respectively, except
3539 that they are delimited by the embedded unquoted equals-sign, in addition to other delimiters.

3540 **Note:** Additional delimiters are described in the Shell and Utilities volume of
3541 IEEE Std. 1003.1-200x, Section 2.3, Token Recognition.

3542 When a variable assignment is done, the variable shall be created if it did not already exist. If
3543 *value* is not specified, the variable shall be given a null value.

3544 **Note:** An alternative form of variable assignment:

3545 *symbol=value*

3546 (where *symbol* is a valid word delimited by an equals-sign, but not a valid name)
3547 produces unspecified results. The form *symbol=value* is used by the KornShell
3548 *name[expression]=value* syntax.



File Format Notation

3550

3551 The STDIN, STDOUT, STDERR, INPUT FILES, and OUTPUT FILES sections of the utility
 3552 descriptions use a syntax to describe the data organization within the files, when that
 3553 organization is not otherwise obvious. The syntax is similar to that used by the System Interfaces
 3554 volume of IEEE Std. 1003.1-200x *printf()* function, as described in this chapter. When used in
 3555 STDIN or INPUT FILES sections of the utility descriptions, this syntax describes the format that
 3556 could have been used to write the text to be read, not a format that could be used by the System
 3557 Interfaces volume of IEEE Std. 1003.1-200x *scanf()* function to read the input file.

3558 The description of an individual record is as follows:

3559 "*format*", [*arg1*, *arg2*, . . . , *argn*]

3560 The *format* is a character string that contains three types of objects defined below:

- 3561 1. *Characters* that are not *escape sequences* or *conversion specifications*, as described below, shall
 3562 be copied to the output.
- 3563 2. *Escape Sequences* represent non-graphic characters.
- 3564 3. *Conversion Specifications* specify the output format of each argument; (see below).

3565 The following characters have the following special meaning in the format string:

3566 ' ' (An empty character position.) Represents one or more <blank> characters.

3567 Δ Represents exactly one <space> character.

3568 Table 5-1 lists escape sequences and associated actions on display devices capable of the action.

3569

Table 5-1 Escape Sequences and Associated Actions

3570

3571

3572

3573

3574

3575

3576

3577

3578

3579

3580

3581

3582

3583

3584

3585

Escape Sequence	Represents Character	Terminal Action
'\\'	backslash	Print the character '\\ '.
'\a'	alert	Attempts to alert the user through audible or visible notification.
'\b'	backspace	Moves the printing position to one column before the current position, unless the current position is the start of a line.
'\f'	form-feed	Moves the printing position to the initial printing position of the next logical page.
'\n'	newline	Moves the printing position to the start of the next line.
'\r'	carriage-return	Moves the printing position to the start of the current line.
'\t'	tab	Moves 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	Moves 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.

3586

3587

Each conversion specification shall be introduced by the percent-sign character ('%'). After the character '%', the following shall appear in sequence:

3588

3589

flags Zero or more *flags*, in any order, that modify the meaning of the conversion specification.

3590

3591

3592

3593

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.

3594

3595

3596

3597

3598

3599

precision Gives the minimum number of digits to appear for the *d*, *o*, *i*, *u*, *x*, or *X* conversions (the field is padded with leading zeros), the number of digits to appear after the radix character for the *e* and *f* conversions, the maximum number of significant digits for the *g* conversion; or the maximum number of bytes to be written from a string in *s* conversion. The precision shall take the form of a period ('.') followed by a decimal digit string; a null digit string is treated as zero.

3600

3601

3602

conversion characters

A conversion character (see below) that indicates the type of conversion to be applied.

3603

The *flag* characters and their meanings are:

3604

3605

3606

3607

3608

- 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> character shall be prefixed to the result. This means that if the <space> character and '+' flags both appear, the <space> character flag shall be ignored.

3609

3610

3611

3612

3613

The value is to be converted to an alternative form. For *c*, *d*, *i*, *u*, and *s* conversions, the behavior is undefined. For *o* conversion, it shall increase the precision to force the first digit of the result to be a zero. For *x* or *X* conversion, a non-zero result has 0x or 0X prefixed to it, respectively. For *e*, *E*, *f*, *g*, and *G* conversions, the result shall always contain a radix character, even if no digits follow the radix character. For *g*

3614		and <i>G</i> conversions, trailing zeros shall not be removed from the result as they
3615		usually are.
3616	0	For <i>d</i> , <i>i</i> , <i>o</i> , <i>u</i> , <i>x</i> , <i>X</i> , <i>e</i> , <i>E</i> , <i>f</i> , <i>g</i> , and <i>G</i> conversions, leading zeros (following any
3617		indication of sign or base) shall be used to pad to the field width; no space padding
3618		is performed. If the '0' and '-' flags both appear, the '0' flag shall be ignored.
3619		For <i>d</i> , <i>i</i> , <i>o</i> , <i>u</i> , <i>x</i> , and <i>X</i> conversions, if a precision is specified, the '0' flag shall be
3620		ignored. For other conversions, the behavior is undefined.
3621		Each conversion character shall result in fetching zero or more arguments. The results are
3622		undefined if there are insufficient arguments for the format. If the format is exhausted while
3623		arguments remain, the excess arguments shall be ignored.
3624		The <i>conversion characters</i> and their meanings are:
3625	<i>d,i,o,u,x,X</i>	The integer argument shall be written as signed decimal (<i>d</i> or <i>i</i>), unsigned octal (<i>o</i>),
3626		unsigned decimal (<i>u</i>), or unsigned hexadecimal notation (<i>x</i> and <i>X</i>). The <i>d</i> and <i>i</i>
3627		specifiers shall convert to signed decimal in the style [-]dddd. The <i>x</i> conversion
3628		shall use the numbers and letters <i>0123456789abcdef</i> and the <i>X</i> conversion shall use
3629		the numbers and letters <i>0123456789ABCDEF</i> . The <i>precision</i> component of the
3630		argument shall specify the minimum number of digits to appear. If the value being
3631		converted can be represented in fewer digits than the specified minimum, it shall
3632		be expanded with leading zeros. The default precision shall be 1. The result of
3633		converting a zero value with a precision of 0 shall be no characters. If both the field
3634		width and precision are omitted, the implementation may precede, follow, or
3635		precede and follow numeric arguments of types <i>d</i> , <i>i</i> , and <i>u</i> with <blank>
3636		characters; arguments of type <i>o</i> (octal) may be preceded with leading zeros.
3637	<i>f</i>	The floating point number argument shall be written in decimal notation in the
3638		style [-]ddd.ddd, where the number of digits after the radix character (shown here
3639		as a decimal point) shall be equal to the <i>precision</i> specification. The <i>LC_NUMERIC</i>
3640		locale category shall determine the radix character to use in this format. If the
3641		<i>precision</i> is omitted from the argument, six digits shall be written after the radix
3642		character; if the <i>precision</i> is explicitly 0, no radix character shall appear.
3643	<i>e,E</i>	The floating point number argument shall be written in the style [-]d.ddd±dd (the
3644		symbol '±' indicates either a plus or minus sign), where there is one digit before
3645		the radix character (shown here as a decimal point) and the number of digits after
3646		it is equal to the precision. The <i>LC_NUMERIC</i> locale category shall determine the
3647		radix character to use in this format. When the precision is missing, six digits shall
3648		be written after the radix character; if the precision is 0, no radix character shall
3649		appear. The <i>E</i> conversion character shall produce a number with <i>E</i> instead of <i>e</i>
3650		introducing the exponent. The exponent shall always contain at least two digits.
3651		However, if the value to be written requires an exponent greater than two digits,
3652		additional exponent digits shall be written as necessary.
3653	<i>g,G</i>	The floating point number argument shall be written in style <i>f</i> or <i>e</i> (or in style <i>E</i> in
3654		the case of a <i>G</i> conversion character), with the precision specifying the number of
3655		significant digits. The style used depends on the value converted: style <i>e</i> (or <i>E</i>)
3656		shall be used only if the exponent resulting from the conversion is less than -4 or
3657		greater than or equal to the precision. Trailing zeros are removed from the result. A
3658		radix character shall appear only if it is followed by a digit.
3659	<i>c</i>	The integer argument shall be converted to an unsigned char and the resulting
3660		byte shall be written.

3661 s The argument shall be taken to be a string and bytes from the string shall be
 3662 written until the end of the string or the number of bytes indicated by the *precision*
 3663 specification of the argument is reached. If the precision is omitted from the
 3664 argument, it shall be taken to be infinite, so all bytes up to the end of the string
 3665 shall be written.

3666 % Write a '%' character; no argument is converted.

3667 In no case does a nonexistent or insufficient *field width* cause truncation of a field; if the result of
 3668 a conversion is wider than the field width, the field is simply expanded to contain the conversion
 3669 result. The term *field width* should not be confused with the term *precision* used in the description
 3670 of %s.

3671 Examples

3672 To represent the output of a program that prints a date and time in the form Sunday, July 3,
 3673 10:02, where *weekday* and *month* are strings:

3674 "%s, %s %d, %d: %.2d\n" <weekday>, <month>, <day>, <hour>, <min>

3675 To show 'π' written to 5 decimal places:

3676 "pi = %.5f\n", <value of π>

3677 To show an input file format consisting of five colon-separated fields:

3678 "%s: %s: %s: %s: %s\n", <arg1>, <arg2>, <arg3>, <arg4>, <arg5>

3679

Character Set

3680

3681 6.1 Portable Character Set

3682 Conforming implementations shall support one or more coded character sets. Each supported
 3683 locale shall include the *portable character set*, which is the set of symbolic names for characters in
 3684 Table 6-1. This is used to describe characters within the text of IEEE Std. 1003.1-200x. The first
 3685 eight entries in Table 6-1 are defined in the ISO/IEC 6429:1992 standard and the rest of the
 3686 characters are defined in the ISO/IEC 10646-1:1993 standard.

3687 **Table 6-1** Portable Character Set

3688

3689

3690

3691

3692

3693

3694

3695

3696

3697

3698

3699

3700

3701

3702

3703

3704

3705

3706

3707

3708

3709

3710

3711

3712

3713

3714

3715

3716

3717

3718

3719

3720

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

3721
3722
3723
3724
3725
3726
3727
3728
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769

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

3770

3771

3772

3773

3774

3775

3776

3777

3778

3779

3780

3781

3782

3783

3784

3785

3786

3787

3788

3789

3790

3791

3792

3793

3794

3795

3796

3797

3798

3799

3800

3801

3802

3803

3804

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

3805

3806

3807

IEEE Std. 1003.1-200x uses other character names 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.

3808

3809

3810

3811

Table 6-1 (on page 133) 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.

3812

3813

IEEE Std. 1003.1-200x places only the following requirements on the encoded values of the characters in the portable character set:

3814

3815

3816

- If the encoded values associated with each member of the portable character set are not invariant across all locales supported by the implementation, the results achieved by an application accessing those locales are unspecified.

3817

3818

- The encoded values associated with the digits 0 to 9 shall be such that the value of each character after 0 shall be one greater than the value of the previous character.

- 3819 • A null character, NUL, which has all bits set to zero, shall be in the set of characters.
- 3820 • The encoded values associated with the members of the portable character set are each
3821 represented in a single byte. Moreover, if the value is stored in an object of C-language type
3822 **char**, it is guaranteed to be positive (except the NUL, which is always zero).
- 3823 Conforming implementations shall support certain character and character set attributes, as
3824 defined in Section 7.2 (on page 144).

3825 **6.2 Character Encoding**

3826 The POSIX locale contains the characters in Table 6-1 (on page 133), which have the properties
3827 listed in Section 7.3.1 (on page 147). Implementations may also add other characters. In other
3828 locales, the presence, meaning, and representation of any additional characters is locale-specific.

3829 In locales other than the POSIX locale, a character may have a state-dependent encoding. There
3830 are two types of these encodings:

- 3831 • A single-shift encoding (where each character not in the initial shift state is preceded by a
3832 shift code) can be defined if each shift-code and character sequence is considered a multi-
3833 byte character. This is done using the concatenated-constant format in a character set
3834 description file, as described in Section 6.4 (on page 137). If the implementation supports a
3835 character encoding of this type, all of the standard utilities in the Shell and Utilities volume of
3836 IEEE Std. 1003.1-200x support it. Use of a single-shift encoding with any of the functions in
3837 the System Interfaces volume of IEEE Std. 1003.1-200x that do not specifically mention the
3838 effects of state-dependent encoding is implementation-defined.
- 3839 • A locking-shift encoding (where the state of the character is determined by a shift code that
3840 may affect more than the single character following it) cannot be defined with the current
3841 character set description file format. Use of a locking-shift encoding with any of the standard
3842 utilities in the Shell and Utilities volume of IEEE Std. 1003.1-200x or with any of the functions
3843 in the System Interfaces volume of IEEE Std. 1003.1-200x that do not specifically mention the
3844 effects of state-dependent encoding is implementation-defined.

3845 While in the initial shift state, all characters in the portable character set retain their usual
3846 interpretation and do not alter the shift state. The interpretation for subsequent bytes in the
3847 sequence is a function of the current shift state. A byte with all bits zero is interpreted as the null
3848 character independent of shift state. Thus a byte with all bits zero shall never occur in the second
3849 or subsequent bytes of a character.

3850 The maximum allowable number of bytes in a character in the current locale is indicated by
3851 {MB_CUR_MAX}, defined in the `<stdlib.h>` header and by the `<mb_cur_max>` value in a
3852 character set description file; see Section 6.4 (on page 137). The implementation's maximum
3853 number of bytes in a character is defined by the C-language macro {MB_LEN_MAX}.

3854 6.3 C Language Wide-Character Codes

3855 In the shell, the standard utilities are written so that the encodings of characters are described by
 3856 the locale's *LC_CTYPE* definition (see Section 7.3.1 (on page 147)) and there is no differentiation
 3857 between characters consisting of single octets (8-bit bytes), larger bytes, or multiple bytes.
 3858 However, in the C language, a differentiation is made. To ease the handling of variable length
 3859 characters, the C language has introduced the concept of wide-character codes.

3860 All wide-character codes in a given process consist of an equal number of bits. This is in contrast
 3861 to characters, which can consist of a variable number of bytes. The byte or byte sequence that
 3862 represents a character can also be represented as a wide-character code. Wide-character codes
 3863 thus provide a uniform size for manipulating text data. A wide-character code having all bits
 3864 zero is the null wide-character code (see Section 3.248 (on page 83)), and terminates wide-
 3865 character strings (see Section 3.434 (on page 116)). The wide-character value for each member of
 3866 the Portable Character Set equals its value when used as the lone character in an integer
 3867 character constant. Wide-character codes for other characters are locale and implementation-
 3868 defined. State shift bytes do not have a wide-character code representation.

3869 6.4 Character Set Description File

3870 Implementations shall provide a character set description file for at least one coded character set
 3871 supported by the implementation. These files are referred to elsewhere in IEEE Std. 1003.1-200x
 3872 as *charmap* files. It is implementation-defined whether or not users or applications can provide
 3873 additional character set description files.

3874 IEEE Std. 1003.1-200x does not require that multiple character sets or codesets be supported.
 3875 Although multiple charmap files are supported, it is the responsibility of the implementation to
 3876 provide the file or files; if only one is provided, only that one is accessible using the *localedef*
 3877 utility's *-f* option.

3878 Each character set description file, except those that use the ISO/IEC 10646-1:1993 standard
 3879 position values as the encoding values, shall define characteristics for the coded character set
 3880 and the encoding for the characters specified in Table 6-1 (on page 133), and may define
 3881 encoding for additional characters supported by the implementation. Other information about
 3882 the coded character set may also be in the file. Coded character set character values shall be
 3883 defined using symbolic character names followed by character encoding values.

3884 Each symbolic name specified in Table 6-1 (on page 133) shall be included in the file and shall be
 3885 mapped to a unique encoding value (except for those symbolic names that are shown with
 3886 identical glyphs). If the control characters commonly associated with the symbolic names in the
 3887 following table are supported by the implementation, the symbolic names and their
 3888 corresponding encoding values shall be included in the file. Some of the encodings associated
 3889 with the symbolic names in this table may be the same as characters in the portable character set
 3890 table.

3891 **Table 6-2** Control Character Set

<ACK>	<DC2>	<ENQ>	<FS>	<IS4>	<SOH>
<BEL>	<DC3>	<EOT>	<GS>	<LF>	<STX>
<BS>	<DC4>	<ESC>	<HT>	<NAK>	<SUB>
<CAN>		<ETB>	<IS1>	<RS>	<SYN>
<CR>	<DLE>	<ETX>	<IS2>	<SI>	<US>
<DC1>		<FF>	<IS3>	<SO>	<VT>

3898 The following declarations can precede the character definitions. Each consists of the symbol
3899 shown in the following list, starting in column 1, including the surrounding brackets, followed
3900 by one or more <blank> characters, followed by the value to be assigned to the symbol.

3901 **<code_set_name>** The name of the coded character set for which the character set
3902 description file is defined. The characters of the name shall be taken from
3903 the set of characters with visible glyphs defined in Table 6-1 (on page
3904 133).

3905 **<mb_cur_max>** The maximum number of bytes in a multi-byte character. This defaults to
3906 1.

3907 **<mb_cur_min>** An unsigned positive integer value that defines the minimum number of
3908 XSI bytes in a character for the encoded character set. On XSI-conformant
3909 systems, **<mb_cur_min>** shall always be 1.

3910 **<escape_char>** The escape character used to indicate that the characters following are
3911 interpreted in a special way, as defined later in this section. This defaults
3912 to backslash ('\''), which is the character glyph used in all the following
3913 text and examples, unless otherwise noted.

3914 **<comment_char>** The character that, when placed in column 1 of a charmap line, is used to
3915 indicate that the line is to be ignored. The default character is the number
3916 sign ('#').

3917 The character set mapping definitions shall be all the lines immediately following an identifier
3918 line containing the string "CHARMAP" starting in column 1, and preceding a trailer line
3919 containing the string "END CHARMAP" starting in column 1. Empty lines and lines containing a
3920 **<comment_char>** in the first column shall be ignored. Each non-comment line of the character
3921 set mapping definition (that is, between the "CHARMAP" and "END CHARMAP" lines of the file)
3922 shall be in either of two forms:

3923 " %s %s %s\n", <symbolic-name>, <encoding>, <comments>

3924 or:

3925 " %s...%s %s %s\n", <symbolic-name>, <symbolic-name>,
3926 <encoding>, <comments>

3927 In the first format, the line in the character set mapping definition defines a single symbolic
3928 name and a corresponding encoding. A symbolic name is one or more characters from the set
3929 shown with visible glyphs in Table 6-1 (on page 133), enclosed between angle brackets. A
3930 character following an escape character is interpreted as itself; for example, the sequence
3931 "<\\>" represents the symbolic name ">" enclosed between angle brackets.

3932 In the second format, the line in the character set mapping definition defines a range of one or
3933 more symbolic names. In this form, the symbolic names shall consist of zero or more non-
3934 numeric characters from the set shown with visible glyphs in Table 6-1 (on page 133), followed
3935 by an integer formed by one or more decimal digits. Both integers shall contain the same number
3936 of digits. The characters preceding the integer shall be identical in the two symbolic names, and
3937 the integer formed by the digits in the second symbolic name shall be equal to or greater than the
3938 integer formed by the digits in the first name. This shall be interpreted as a series of symbolic
3939 names formed from the common part and each of the integers between the first and the second
3940 integer, inclusive. As an example, <j0101>...<j0104> is interpreted as the symbolic names
3941 <j0101>, <j0102>, <j0103>, and <j0104>, in that order.

3942 A character set mapping definition line shall exist for all symbolic names specified in Table 6-1
3943 (on page 133), and defines the coded character value that corresponds to the character glyph

3944 indicated in the table, or the coded character value that corresponds to the control character
 3945 symbolic name. If the control characters commonly associated with the symbolic names in Table
 3946 6-2 (on page 137) are supported by the implementation, the symbolic name and the
 3947 corresponding encoding value shall be included in the file. Additional unique symbolic names
 3948 may be included. A coded character value can be represented by more than one symbolic name.

3949 The encoding part is expressed as one (for single-byte character values) or more concatenated
 3950 decimal, octal, or hexadecimal constants in the following formats:

```
3951     "%cd%u", <escape_char>, <decimal byte value>
3952     "%cx%x", <escape_char>, <hexadecimal byte value>
3953     "%co", <escape_char>, <octal byte value>
```

3954 Decimal constants are represented by two or three decimal digits, preceded by the escape
 3955 character and the lowercase letter 'd'; for example, "\d05", "\d97", or "\d143".
 3956 Hexadecimal constants are represented by two hexadecimal digits, preceded by the escape
 3957 character and the lowercase letter 'x'; for example, "\x05", "\x61", or "\x8f". Octal
 3958 constants are represented by two or three octal digits, preceded by the escape character; for
 3959 example, "\05", "\141", or "\217". In a portable charmap file, each constant represents an 8-
 3960 bit byte. Implementations supporting other byte sizes may allow constants to represent values
 3961 larger than those that can be represented in 8-bit bytes, and to allow additional digits in
 3962 constants. When constants are concatenated for multi-byte character values, they shall be of the
 3963 same type, and interpreted in byte order from first to last with the least significant byte of the
 3964 multi-byte character specified by the last constant. The manner in which these constants are
 3965 represented in the character stored in the system is implementation-defined. (This notation was
 3966 chosen for reasons of portability. There is no requirement that the internal representation in the
 3967 computer memory be in this same order.) Omitting bytes from a multi-byte character definition
 3968 produces undefined results.

3969 In lines defining ranges of symbolic names, the encoded value is the value for the first symbolic
 3970 name in the range (the symbolic name preceding the ellipsis). Subsequent symbolic names
 3971 defined by the range shall have encoding values in increasing order. For example, the line:

```
3972     <j0101>...<j0104> \d129\d254
```

3973 is interpreted as:

```
3974     <j0101>          \d129\d254
3975     <j0102>          \d129\d255
3976     <j0103>          \d130\d0
3977     <j0104>          \d130\d1
```

3978 Note that this line is interpreted as the example even on systems with bytes larger than 8 bits.

3979 In lines defining ranges of symbolic names that also use the ISO/IEC 10646-1:1993 standard
 3980 position constant values, the conversion to the target codeset encoding value shall be performed
 3981 before assignment of encoding values to symbolic names.

3982 The comment is optional.

3983 The following declarations can follow the character set mapping definitions (after the "END
 3984 CHARMAP" statement). Each shall consist of the keyword shown in the following list, starting in
 3985 column 1, followed by the value(s) to be associated to the keyword, as defined below.

3986 **WIDTH** An unsigned positive integer value defining the column width (see Section 3.106
 3987 (on page 59)) for the printable characters in the coded character set specified in
 3988 Table 6-1 (on page 133) and Table 6-2 (on page 137).

3989 **Notes to Reviewers**3990 *This section with side shading will not appear in the final copy. - Ed.*

3991 D3, XBD, ERN 90 suggests alternative wording for the text above: "the printable
3992 characters specified between the CHARMAP and END CHARMAP statements".
3993 The current wording is as per P1003.2b. When .2b is approved, an interpretation
3994 should be filed.

3995 Coded character set character values shall be defined using symbolic character
3996 names followed by column width values. Defining a character with more than one
3997 **WIDTH** produces undefined results. The **END WIDTH** keyword shall be used to
3998 terminate the **WIDTH** definitions. Specifying the width of a non-printable
3999 character in a **WIDTH** declaration produces undefined results.

4000 **WIDTH_DEFAULT**

4001 An unsigned positive integer value defining the default column width for any
4002 printable character not listed by one of the **WIDTH** keywords. If no
4003 **WIDTH_DEFAULT** keyword is included in the charmap, the default character
4004 width shall be 1.

4005 **Example**

4006 After the "END CHARMAP" statement, a syntax for a width definition would be:

```
4007 WIDTH
4008 <A> 1
4009 <B> 1
4010 <C>...<Z> 1
4011 <fool>...<foon> 2
4012 END WIDTH
```

4013 In this example, the numerical code point values represented by the symbols <A> and are
4014 assigned a width of 1. The code point values <C> to <Z> inclusive (<C>, <D>, <E>, and so on)
4015 are also assigned a width of 1. Using <A>...<Z> would have required fewer lines, but the
4016 alternative was shown to demonstrate flexibility. The keyword **WIDTH_DEFAULT** could have
4017 been added as appropriate.

4018 **6.4.1 State-Dependent Character Encodings**

4019 This section addresses the use of state-dependent character encodings (that is, those in which the
4020 encoding of a character is dependent on one or more shift codes that may precede it).

4021 A single-shift encoding (where each character not in the initial shift state is preceded by a shift
4022 code) can be defined in the charmap format if each shift-code/character sequence is considered a
4023 multi-byte character, defined using the concatenated-constant format described in Section 6.4
4024 (on page 137). If the implementation supports a character encoding of this type, all of the
4025 standard utilities shall support it. A locking-shift encoding (where the state of the character is
4026 determined by a shift code that may affect more than the single character following it) could be
4027 defined with an extension to the charmap format described in Section 6.4 (on page 137). If the
4028 implementation supports a character encoding of this type, any of the standard utilities that
4029 describe character (*versus* byte) or text-file manipulation shall have the following characteristics:

- 4030 1. The utility shall process the statefully encoded data as a concatenation of state-
4031 independent characters. The presence of redundant locking shifts shall not affect the
4032 comparison of two statefully encoded strings.

- 4033 2. A utility that divides, truncates, or extracts substrings from statefully encoded data shall
4034 produce output that contains locking shifts at the beginning or end of the resulting data, if
4035 appropriate, to retain correct state information.



4037

4038 **7.1 General**

4039 A *locale* is the definition of the subset of a user's environment that depends on language and
 4040 cultural conventions. It is made up from one or more categories. Each category is identified by
 4041 its name and controls specific aspects of the behavior of components of the system. Category
 4042 names correspond to the following environment variable names:

4043 *LC_CTYPE* Character classification and case conversion.

4044 *LC_COLLATE* Collation order.

4045 *LC_TIME* Date and time formats.

4046 *LC_NUMERIC* Numeric, non-monetary formatting.

4047 *LC_MONETARY* Monetary formatting.

4048 *LC_MESSAGES* Formats of informative and diagnostic messages and interactive responses.

4049 The standard utilities in the Shell and Utilities volume of IEEE Std. 1003.1-200x shall base their
 4050 behavior on the current locale, as defined in the ENVIRONMENT VARIABLES section for each
 4051 utility. The behavior of some of the C-language functions defined in the System Interfaces
 4052 volume of IEEE Std. 1003.1-200x shall also be modified based on the current locale, as defined by
 4053 the last call to *setlocale()*.

4054 Locales other than those supplied by the implementation can be created via the *localedef* utility,
 4055 provided that the *_POSIX2_LOCALEDEF* symbol is defined on the system. Even if *localedef* is not
 4056 provided, all implementations conforming to the System Interfaces volume of
 4057 IEEE Std. 1003.1-200x shall provide one or more locales that behave as described in this chapter.
 4058 The input to the utility is described in Section 7.3 (on page 145). The value that is used to specify
 4059 a locale when using environment variables shall be the string specified as the *name* operand to
 4060 the *localedef* utility when the locale was created. The strings "C" and "POSIX" are reserved as
 4061 identifiers for the POSIX locale (see Section 7.2 (on page 144)). When the value of a locale
 4062 environment variable begins with a slash ('/'), it is interpreted as the path name of the locale
 4063 definition; the type of file (regular, directory, and so on) used to store the locale definition is
 4064 implementation-defined. If the value does not begin with a slash, the mechanism used to locate
 4065 the locale is implementation-defined.

4066 If different character sets are used by the locale categories, the results achieved by an application
 4067 utilizing these categories are undefined. Likewise, if different codesets are used for the data
 4068 being processed by interfaces whose behavior is dependent on the current locale, or the codeset
 4069 is different from the codeset assumed when the locale was created, the result is also undefined.

4070 Applications can select the desired locale by invoking the *setlocale()* function (or equivalent)
 4071 with the appropriate value. If the function is invoked with an empty string, such as:

```
4072     setlocale(LC_ALL, "");
```

4073 the value of the corresponding environment variable is used. If the environment variable is
 4074 unset or is set to the empty string, the implementation shall set the appropriate environment as
 4075 defined in Chapter 8 (on page 187).

4076 **7.2 POSIX Locale**

4077 Conforming systems shall provide a *POSIX locale*, also known as the C locale. The behavior of
4078 standard utilities and functions in the POSIX locale shall be as if the locale was defined via the
4079 *localedef* utility with input data from the POSIX locale tables in Section 7.3 (on page 145).

4080 The tables in Section 7.3 (on page 145) describe the characteristics and behavior of the POSIX
4081 locale for data consisting entirely of characters from the portable character set and the control
4082 character set. For other characters, the behavior is unspecified. For C-language programs, the
4083 POSIX locale is the default locale when the *setlocale()* function is not called.

4084 The POSIX locale can be specified by assigning to the appropriate environment variables the
4085 values "C" or "POSIX".

4086 All implementations shall define a locale as the default locale, to be invoked when no
4087 environment variables are set, or set to the empty string. This default locale can be the POSIX
4088 locale or any other implementation-defined locale. Some implementations may provide facilities |
4089 for local installation administrators to set the default locale, customizing it for each location. |
4090 IEEE Std. 1003.1-200x does not require such a facility.

4091 **7.3 Locale Definition**

4092 The capability to specify additional locales to those provided by an implementation is optional,
 4093 denoted by the `_POSIX2_LOCALEDEF` symbol. If the option is not supported, only
 4094 implementation-supplied locales are available.

4095 Locales can be described with the file format presented in this section. The file format is that
 4096 accepted by the *localedef* utility. For the purposes of this section, the file is referred to as the *locale*
 4097 *definition file*, but no locales shall be affected by this file unless it is processed by *localedef* or some
 4098 similar mechanism. Any requirements in this section imposed upon the utility shall apply to
 4099 *localedef* or to any other similar utility used to install locale information using the locale
 4100 definition file format described here.

4101 The locale definition file shall contain one or more locale category source definitions, and shall
 4102 not contain more than one definition for the same locale category. If the file contains source
 4103 definitions for more than one category, implementation-defined categories, if present, shall
 4104 appear after the categories defined by Section 7.1 (on page 143). A category source definition
 4105 contains either the definition of a category or a **copy** directive. For a description of the **copy**
 4106 directive, see *localedef*. In the event that some of the information for a locale category, as
 4107 specified in this volume of IEEE Std. 1003.1-200x, is missing from the locale source definition, the
 4108 behavior of that category, if it is referenced, is unspecified.

4109 A category source definition consists of a category header, a category body, and a category
 4110 trailer. A category header consists of the character string naming of the category, beginning with
 4111 the characters *LC_*. The category trailer consists of the string "END", followed by one or more
 4112 <blank> characters and the string used in the corresponding category header.

4113 The category body consists of one or more lines of text. Each line contains an identifier,
 4114 optionally followed by one or more operands. Identifiers are either keywords, identifying a
 4115 particular locale element, or collating elements. In addition to the keywords defined in this
 4116 volume of IEEE Std. 1003.1-200x, the source can contain implementation-defined keywords.
 4117 Each keyword within a locale has a unique name (that is, two categories cannot have a
 4118 commonly-named keyword); no keyword can start with the characters *LC_*. Identifiers are
 4119 separated from the operands by one or more <blank> characters.

4120 Operands shall be characters, collating elements, or strings of characters. Strings are enclosed in
 4121 double-quotes. Literal double-quotes within strings are preceded by the <escape character>,
 4122 described below. When a keyword is followed by more than one operand, the operands are
 4123 separated by semicolons; <blank> characters are allowed both before and after a semicolon.

4124 The first category header in the file can be preceded by a line modifying the comment character.
 4125 It shall have the following format, starting in column 1:

```
4126 "comment_char %c\n", <comment character>
```

4127 The comment character defaults to the number sign ('#'). Blank lines and lines containing the
 4128 <comment character> in the first position are ignored.

4129 The first category header in the file can be preceded by a line modifying the escape character to
 4130 be used in the file. It shall have the following format, starting in column 1:

```
4131 "escape_char %c\n", <escape character>
```

4132 The escape character defaults to backslash, which is the character used in all examples shown in
 4133 this volume of IEEE Std. 1003.1-200x.

4134 A line can be continued by placing an escape character as the last character on the line; this
 4135 continuation character is discarded from the input. Although the implementation need not
 4136 accept any one portion of a continued line with a length exceeding {LINE_MAX} bytes, it shall

4137 place no limits on the accumulated length of the continued line. Comment lines cannot be
4138 continued on a subsequent line using an escaped newline character.

4139 Individual characters, characters in strings, and collating elements shall be represented using
4140 symbolic names, as defined below. In addition, characters can be represented using the
4141 characters themselves or as octal, hexadecimal, or decimal constants. When non-symbolic
4142 notation is used, the resultant locale definitions are in many cases not portable between systems.
4143 The left angle bracket ('<') is a reserved symbol, denoting the start of a symbolic name; when
4144 used to represent itself it shall be preceded by the escape character. The following rules apply to
4145 character representation:

4146 1. A character can be represented via a symbolic name, enclosed within angle brackets '<'
4147 and '>'. The symbolic name, including the angle brackets, shall exactly match a symbolic
4148 name defined in the charmap file specified via the *localedef* -f option, and it shall be
4149 replaced by a character value determined from the value associated with the symbolic
4150 name in the charmap file. The use of a symbolic name not found in the charmap file shall
4151 constitute an error, unless the category is *LC_CTYPE* or *LC_COLLATE*, in which case it
4152 shall constitute a warning condition (see *localedef* for a description of action resulting from
4153 errors and warnings). The specification of a symbolic name in a **collating-element** or
4154 **collating-symbol** section that duplicates a symbolic name in the charmap file (if present)
4155 shall be an error. Use of the escape character or a right angle bracket within a symbolic
4156 name is invalid unless the character is preceded by the escape character.

4157 For example:

```
4158 <c>;<c-cedilla> " <M><a><y>"
```

4159 2. A character in the portable character set can be represented by the character itself, in which
4160 case the value of the character is implementation-defined. (Implementations may allow
4161 other characters to be represented as themselves, but such locale definitions are not
4162 portable.) Within a string, the double-quote character, the escape character, and the right
4163 angle bracket character shall be escaped (preceded by the escape character) to be
4164 interpreted as the character itself. Outside strings, the characters:

```
4165 , ; < > escape_char
```

4166 shall be escaped to be interpreted as the character itself.

4167 For example:

```
4168 c β "May"
```

4169 3. A character can be represented as an octal constant. An octal constant is specified as the
4170 escape character followed by two or three octal digits. Each constant represents a byte
4171 value. Multi-byte values can be represented by concatenated constants specified in byte
4172 order with the last constant specifying the least significant byte of the character.

4173 For example:

```
4174 \143;\347;\143\150 "\115\141\171"
```

4175 4. A character can be represented as a hexadecimal constant. A hexadecimal constant shall be
4176 specified as the escape character followed by an 'x' followed by two hexadecimal digits.
4177 Each constant shall represent a byte value. Multi-byte values can be represented by
4178 concatenated constants specified in byte order with the last constant specifying the least
4179 significant byte of the character.

4180 For example:

4181 \0x63; \0xe7; \0x63\0x68 "\0x4d\0x61\0x79"

4182 5. A character can be represented as a decimal constant. A decimal constant shall be specified
4183 as the escape character followed by a 'd' followed by two or three decimal digits. Each
4184 constant represents a byte value. Multi-byte values can be represented by concatenated
4185 constants specified in byte order with the last constant specifying the least significant byte
4186 of the character.

4187 For example:

4188 \0d99; \0d231; \0d99\0d104 "\0d77\0d97\0d121"

4189 Implementations supporting other byte sizes may allow constants to represent values larger
4190 than those that can be represented in 8-bit bytes, and to allow additional digits in constants.

4191 Implementations may accept single-digit octal, decimal, or hexadecimal constants following the
4192 escape character. Only characters existing in the character set for which the locale definition is
4193 created can be specified, whether using symbolic names, the characters themselves, or octal,
4194 decimal, or hexadecimal constants. If a charmap file is present, only characters defined in the
4195 charmap can be specified using octal, decimal, or hexadecimal constants. Symbolic names not
4196 present in the charmap file can be specified and shall be ignored, as specified under item 1
4197 above.

4198 7.3.1 LC_CTYPE

4199 The *LC_CTYPE* category shall define character classification, case conversion, and other
4200 character attributes. In addition, a series of characters can be represented by three adjacent
4201 periods representing an ellipsis symbol ("..."). The ellipsis specification shall be interpreted as
4202 meaning that all values between the values preceding and following it represent valid
4203 characters. The ellipsis specification is valid only within a single encoded character set; that is,
4204 within a group of characters of the same size. An ellipsis shall be interpreted as including in the
4205 list all characters with an encoded value higher than the encoded value of the character
4206 preceding the ellipsis and lower than the encoded value of the character following the ellipsis.

4207 For example:

4208 \0x30; ...; \0x39;

4209 includes in the character class all characters with encoded values between the endpoints.

4210 The following keywords shall be recognized. In the descriptions, the term “automatically
4211 included” means that it shall not be an error either to include or omit any of the referenced
4212 characters; the implementation provides them if missing (even if the entire keyword is missing)
4213 and accepts them silently if present. When the implementation automatically includes a missing
4214 character, it shall have an encoded value dependent on the charmap file in effect (see the
4215 description of the *localedef -f* option); otherwise, it has a value derived from an implementation-
4216 defined character mapping.

4217 The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included
4218 characters. These only need to be specified if the character values (that is, encoding) differ from
4219 the implementation default values. It is not possible to define a locale without these
4220 automatically included characters unless some implementation extension is used to prevent
4221 their inclusion. Such a definition would not be a proper superset of the C or POSIX locale and
4222 thus, it might not be possible for conforming applications to work properly.

4223 **copy** Specify the name of an existing locale to be used as the definition of this
4224 category. If this keyword is specified, no other keyword can be specified.

4225	upper	Define characters to be classified as uppercase letters.
4226		In the POSIX locale, the 26 uppercase letters are included:
4227		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
4228		In a locale definition file, no character specified for the keywords cntrl , digit ,
4229		punct , or space can be specified. The uppercase letters <A> to <Z>, as defined
4230		in Section 6.4 (on page 137) (the portable character set), are automatically
4231		included in this class.
4232	lower	Define characters to be classified as lowercase letters.
4233		In the POSIX locale, the 26 lowercase letters are included:
4234		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
4235		In a locale definition file, no character specified for the keywords cntrl , digit ,
4236		punct , or space can be specified. The lowercase letters <a> to <z> of the
4237		portable character set are automatically included in this class.
4238	alpha	Define characters to be classified as letters.
4239		In the POSIX locale, all characters in the classes upper and lower are included.
4240		In a locale definition file, no character specified for the keywords cntrl , digit ,
4241		punct , or space can be specified. Characters classified as either upper or lower
4242		are automatically included in this class.
4243	digit	Define the characters to be classified as numeric digits.
4244		In the POSIX locale, only:
4245		0 1 2 3 4 5 6 7 8 9
4246		are included.
4247		In a locale definition file, only the digits <zero>, <one>, <two>, <three>,
4248		<four>, <five>, <six>, <seven>, <eight>, and <nine> can be specified, and in
4249		contiguous ascending sequence by numerical value. The digits <zero> to
4250		<nine> of the portable character set are automatically included in this class.
4251		The definition of character class digit requires that only ten characters—the
4252		ones defining digits—can be specified; alternative digits (for example, Hindi
4253		or Kanji) cannot be specified here. However, the encoding may vary if an
4254		implementation supports more than one encoding.
4255	alnum	Define characters to be classified as letters and numeric digits. Only the
4256		characters specified for the alpha and digit keywords shall be specified.
4257		Characters specified for the keywords alpha and digit are automatically
4258		included in this class.
4259	space	Define characters to be classified as white-space characters.
4260		In the POSIX locale, at a minimum, the characters <space>, <form-feed>,
4261		<newline>, <carriage-return>, <tab>, and <vertical-tab> are included.
4262		In a locale definition file, no character specified for the keywords upper ,
4263		lower , alpha , digit , graph , or xdigit can be specified. The characters <space>,
4264		<form-feed>, <newline>, <carriage-return>, <tab>, and <vertical-tab> of the
4265		portable character set, and any characters included in the class blank are
4266		automatically included in this class.

4267	cntrl	Define characters to be classified as control characters.
4268		In the POSIX locale, no characters in classes alpha or print are included.
4269		In a locale definition file, no character specified for the keywords upper ,
4270		lower , alpha , digit , punct , graph , print , or xdigit can be specified.
4271	punct	Define characters to be classified as punctuation characters.
4272		In the POSIX locale, neither the <space> character nor any characters in
4273		classes alpha , digit , or cntrl are included.
4274		In a locale definition file, no character specified for the keywords upper ,
4275		lower , alpha , digit , cntrl , xdigit , or as the <space> character can be specified.
4276	graph	Define characters to be classified as printable characters, not including the
4277		<space> character.
4278		In the POSIX locale, all characters in classes alpha , digit , and punct are
4279		included; no characters in class cntrl are included.
4280		In a locale definition file, characters specified for the keywords upper , lower ,
4281		alpha , digit , xdigit , and punct are automatically included in this class. No
4282		character specified for the keyword cntrl can be specified.
4283	print	Define characters to be classified as printable characters, including the
4284		<space> character.
4285		In the POSIX locale, all characters in class graph are included; no characters in
4286		class cntrl are included.
4287		In a locale definition file, characters specified for the keywords upper , lower ,
4288		alpha , digit , xdigit , punct , graph , and the <space> character are automatically
4289		included in this class. No character specified for the keyword cntrl can be
4290		specified.
4291	xdigit	Define the characters to be classified as hexadecimal digits.
4292		In the POSIX locale, only:
4293		0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f
4294		are included.
4295		In a locale definition file, only the characters defined for the class digit can be
4296		specified, in contiguous ascending sequence by numerical value, followed by
4297		one or more sets of six characters representing the hexadecimal digits 10 to 15
4298		inclusive, with each set in ascending order (for example, <A>, , <C>, <D>,
4299		<E>, <F>, <a>, , <c>, <d>, <e>, <f>). The digits <zero> to <nine>, the
4300		uppercase letters <A> to <F>, and the lowercase letters <a> to <f> of the
4301		portable character set are automatically included in this class.
4302		The definition of character class xdigit requires that the characters included in
4303		character class digit be included here also.
4304	blank	Define characters to be classified as <blank> characters.
4305		In the POSIX locale, only the <space> and <tab> characters are included.
4306		In a locale definition file, the characters <space> and <tab> are automatically
4307		included in this class.

4308 **charclass** Define one or more locale-specific character class names as strings separated
4309 by semicolons. Each named character class can then be defined subsequently
4310 in the *LC_CTYPE* definition. A character class name consists of at least one
4311 and at most {CHARCLASS_NAME_MAX} bytes of alphanumeric characters
4312 from the portable file name character set. The first character of a character
4313 class name cannot be a digit. The name cannot match any of the *LC_CTYPE*
4314 keywords defined in this volume of IEEE Std. 1003.1-200x. Future revisions of
4315 IEEE Std. 1003.1-200x will not specify any *LC_CTYPE* keywords containing
4316 uppercase letters.

4317 *charclass-name* Define characters to be classified as belonging to the named locale-specific
4318 character class. In the POSIX locale, the locale-specific named character classes
4319 need not exist.

4320 If a class name is defined by a **charclass** keyword, but no characters are
4321 subsequently assigned to it, this is not an error; it represents a class without
4322 any characters belonging to it.

4323 The *charclass-name* can be used as the *property* argument to the *wctype()*
4324 function, in regular expression and shell pattern-matching bracket
4325 expressions, and by the *tr* command.

4326 **toupper** Define the mapping of lowercase letters to uppercase letters.

4327 In the POSIX locale, at a minimum, the 26 lowercase characters:

4328 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

4329 are mapped to the corresponding 26 uppercase characters:

4330 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

4331 In a locale definition file, the operand consists of character pairs, separated by
4332 semicolons. The characters in each character pair are separated by a comma
4333 and the pair enclosed by parentheses. The first character in each pair is the
4334 lowercase letter, the second the corresponding uppercase letter. Only
4335 characters specified for the keywords **lower** and **upper** can be specified. The
4336 lowercase letters <a> to <z>, and their corresponding uppercase letters <A> to
4337 <Z>, of the portable character set are automatically included in this mapping,
4338 but only when the **toupper** keyword is omitted from the locale definition.

4339 **tolower** Define the mapping of uppercase letters to lowercase letters.

4340 In the POSIX locale, at a minimum, the 26 uppercase characters:

4341 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

4342 are mapped to the corresponding 26 lowercase characters:

4343 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

4344 In a locale definition file, the operand consists of character pairs, separated by
4345 semicolons. The characters in each character pair are separated by a comma
4346 and the pair enclosed by parentheses. The first character in each pair is the
4347 uppercase letter, the second the corresponding lowercase letter. Only
4348 characters specified for the keywords **lower** and **upper** can be specified. If the
4349 **tolower** keyword is omitted from the locale definition, the mapping is the
4350 reverse mapping of the one specified for **toupper**.

4351 The following table shows the character class combinations allowed:

4352 **Table 7-1 Valid Character Class Combinations**

4353 4354 In Class	4353 Can Also Belong To										
	upper	lower	alpha	digit	space	cntrl	punct	graph	print	xdigit	blank
4355 upper	—	—	A	x	x	x	x	A	A	—	x
4356 lower	—	—	A	x	x	x	x	A	A	—	x
4357 alpha	—	—	—	x	x	x	x	A	A	—	x
4358 digit	x	x	x	—	x	x	x	A	A	A	x
4359 space	x	x	x	x	—	—	*	*	*	x	—
4360 cntrl	x	x	x	x	—	—	x	x	x	x	—
4361 punct	x	x	x	x	—	x	—	A	A	x	—
4362 graph	—	—	—	—	—	x	—	—	A	—	—
4363 print	—	—	—	—	—	x	—	—	—	—	—
4364 xdigit	—	—	—	—	x	x	x	A	A	—	x
4365 blank	x	x	x	x	A	—	*	*	*	x	—

4366 **Notes:**

- 4367 1. Explanation of codes:
- 4368 A Automatically included; see text.
- 4369 — Permitted.
- 4370 x Mutually-exclusive.
- 4371 * See note 2.
- 4372 2. The <space> character, which is part of the **space** and **blank** classes, cannot
- 4373 belong to **punct** or **graph**, but automatically belongs to the **print** class. Other
- 4374 **space** or **blank** characters can be classified as any of **punct**, **graph**, or **print**.

4375 The character classifications for the POSIX locale follow; the code listing depicting the *localedef*

4376 input, the table representing the same information, sorted by character.

```

4377 LC_CTYPE
4378 # The following is the POSIX locale LC_CTYPE.
4379 # "alpha" is by default "upper" and "lower"
4380 # "alnum" is by definition "alpha" and "digit"
4381 # "print" is by default "alnum", "punct" and the <space> character
4382 # "graph" is by default "alnum" and "punct"
4383 #
4384 upper <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;<M>;\
4385 <N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;<Z>
4386 #
4387 lower <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
4388 <n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>
4389 #
4390 digit <zero>;<one>;<two>;<three>;<four>;<five>;<six>;\
4391 <seven>;<eight>;<nine>
4392 #
4393 space <tab>;<newline>;<vertical-tab>;<form-feed>;\
4394 <carriage-return>;<space>
4395 #
4396 cntrl <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;\

```

```

4397         <form-feed>;<carriage-return>;\
4398         <NUL>;<SOH>;<STX>;<ETX>;<EOT>;<ENQ>;<ACK>;<SO>;\
4399         <SI>;<DLE>;<DC1>;<DC2>;<DC3>;<DC4>;<NAK>;<SYN>;\
4400         <ETB>;<CAN>;<EM>;<SUB>;<ESC>;<IS4>;<IS3>;<IS2>;\
4401         <IS1>;<DEL>
4402     #
4403     punct  <exclamation-mark>;<quotation-mark>;<number-sign>;\
4404            <dollar-sign>;<percent-sign>;<ampersand>;<apostrophe>;\
4405            <left-parenthesis>;<right-parenthesis>;<asterisk>;\
4406            <plus-sign>;<comma>;<hyphen>;<period>;<slash>;\
4407            <colon>;<semicolon>;<less-than-sign>;<equals-sign>;\
4408            <greater-than-sign>;<question-mark>;<commercial-at>;\
4409            <left-square-bracket>;<backslash>;<right-square-bracket>;\
4410            <circumflex>;<underscore>;<grave-accent>;<left-curly-bracket>;\
4411            <vertical-line>;<right-curly-bracket>;<tilde>
4412     #
4413     xdigit  <zero>;<one>;<two>;<three>;<four>;<five>;<six>;<seven>;\
4414            <eight>;<nine>;<A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
4415     #
4416     blank  <space>;<tab>
4417     #
4418     toupper (<a>,<A>);(<b>,<B>);(<c>,<C>);(<d>,<D>);(<e>,<E>);\
4419            (<f>,<F>);(<g>,<G>);(<h>,<H>);(<i>,<I>);(<j>,<J>);\
4420            (<k>,<K>);(<l>,<L>);(<m>,<M>);(<n>,<N>);(<o>,<O>);\
4421            (<p>,<P>);(<q>,<Q>);(<r>,<R>);(<s>,<S>);(<t>,<T>);\
4422            (<u>,<U>);(<v>,<V>);(<w>,<W>);(<x>,<X>);(<y>,<Y>);(<z>,<Z>)
4423     #
4424     tolower (<A>,<a>);(<B>,<b>);(<C>,<c>);(<D>,<d>);(<E>,<e>);\
4425            (<F>,<f>);(<G>,<g>);(<H>,<h>);(<I>,<i>);(<J>,<j>);\
4426            (<K>,<k>);(<L>,<l>);(<M>,<m>);(<N>,<n>);(<O>,<o>);\
4427            (<P>,<p>);(<Q>,<q>);(<R>,<r>);(<S>,<s>);(<T>,<t>);\
4428            (<U>,<u>);(<V>,<v>);(<W>,<w>);(<X>,<x>);(<Y>,<y>);(<Z>,<z>)
4429     END LC_CTYPE

```


4430
4431
4432
4433
4434
4435
4436
4437
4438
4439
4440
4441
4442
4443
4444
4445
4446
4447
4448
4449
4450
4451
4452
4453
4454
4455
4456
4457
4458
4459
4460
4461
4462
4463
4464
4465
4466
4467
4468
4469
4470
4471
4472
4473
4474
4475
4476
4477
4478

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

4479
4480
4481
4482
4483
4484
4485
4486
4487
4488
4489
4490
4491
4492
4493
4494
4495
4496
4497
4498
4499
4500
4501
4502
4503
4504
4505
4506
4507
4508
4509
4510
4511
4512
4513
4514
4515
4516
4517
4518
4519
4520
4521
4522
4523
4524
4525
4526
4527

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

4528
4529
4530
4531
4532
4533
4534
4535
4536
4537
4538
4539
4540
4541
4542
4543
4544
4545
4546
4547
4548
4549
4550
4551
4552
4553
4554
4555
4556
4557
4558
4559
4560
4561
4562
4563

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

4564 **7.3.2 LC_COLLATE**

4565 The *LC_COLLATE* category provides a collation sequence definition for numerous utilities in the
4566 Shell and Utilities volume of IEEE Std. 1003.1-200x (*sort*, *uniq*, and so on), regular expression
4567 matching (see Chapter 9 (on page 195)) and the *strcoll()*, *strxfrm()*, *wscoll()*, and *wcsxfrm()*
4568 functions in the System Interfaces volume of IEEE Std. 1003.1-200x.

4569 A collation sequence definition shall define the relative order between collating elements
4570 (characters and multi-character collating elements) in the locale. This order is expressed in terms
4571 of collation values; that is, by assigning each element one or more collation values (also known
4572 as collation weights). This does not imply that implementations shall assign such values, but
4573 that ordering of strings using the resultant collation definition in the locale behaves as if such
4574 assignment is done and used in the collation process. At least the following capabilities are
4575 provided:

- 4576 1. **Multi-character collating elements.** Specification of multi-character collating elements
4577 (that is, sequences of two or more characters to be collated as an entity).

- 4578 2. **User-defined ordering of collating elements.** Each collating element shall be assigned a
 4579 collation value defining its order in the character (or basic) collation sequence. This
 4580 ordering is used by regular expressions and pattern matching and, unless collation weights
 4581 are explicitly specified, also as the collation weight to be used in sorting.
- 4582 3. **Multiple weights and equivalence classes.** Collating elements can be assigned one or
 4583 more (up to the limit {COLL_WEIGHTS_MAX}, as defined in <limits.h>) collating weights
 4584 for use in sorting. The first weight is hereafter referred to as the primary weight.
- 4585 4. **One-to-many mapping.** A single character is mapped into a string of collating elements.
- 4586 5. **Equivalence class definition.** Two or more collating elements have the same collation
 4587 value (primary weight).
- 4588 6. **Ordering by weights.** When two strings are compared to determine their relative order,
 4589 the two strings are first broken up into a series of collating elements; the elements in each
 4590 successive pair of elements are then compared according to the relative primary weights
 4591 for the elements. If equal, and more than one weight has been assigned, then the pairs of
 4592 collating elements are recompared according to the relative subsequent weights, until
 4593 either a pair of collating elements compare unequal or the weights are exhausted.

4594 The following keywords shall be recognized in a collation sequence definition. They are
 4595 described in detail in the following sections.

4596	copy	Specify the name of an existing locale to be used as the definition of this category. If this keyword is specified, no other keyword can be specified.
4597		
4598	collating-element	Define a collating-element symbol representing a multi-character collating element. This keyword is optional.
4599		
4600	collating-symbol	Define a collating symbol for use in collation order statements. This keyword is optional.
4601		
4602	order_start	Define collation rules. This statement is followed by one or more collation order statements, assigning character collation values and collation weights to collating elements.
4603		
4604		
4605	order_end	Specify the end of the collation-order statements.

4606 7.3.2.1 *The collating-element Keyword*

4607 In addition to the collating elements in the character set, the **collating-element** keyword can be
 4608 used to define multi-character collating elements. The syntax is as follows:

```
4609     "collating-element %s from \"%s\"\\n", <collating-symbol>, <string>
```

4610 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.

4615 For example:

```
4616     collating-element <ch> from "<c><h>"
4617     collating-element <e-acute> from "<acute><e>"
4618     collating-element <ll> from "ll"
```

4619 7.3.2.2 *The collating-symbol Keyword*

4620 This keyword can be used to define symbols for use in collation sequence statements; that is,
4621 between the **order_start** and the **order_end** keywords. The syntax is as follows:

```
4622 "collating-symbol %s\n", <collating-symbol>
```

4623 The <collating-symbol> shall be a symbolic name, enclosed between angle brackets ('<' and
4624 '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any
4625 other symbolic name defined in this collation definition. A <collating-symbol> defined via this
4626 keyword is only recognized with the *LC_COLLATE* category.

4627 For example:

```
4628 collating-symbol <UPPER_CASE>  
4629 collating-symbol <HIGH>
```

4630 The **collating-symbol** keyword defines a symbolic name that can be associated with a relative
4631 position in the character order sequence. While such a symbolic name does not represent any
4632 collating element, it can be used as a weight.

4633 7.3.2.3 *The order_start Keyword*

4634 The **order_start** keyword shall precede collation order entries and also define the number of
4635 weights for this collation sequence definition and other collation rules.

4636 The syntax of the **order_start** keyword is as follows:

```
4637 "order_start %s;%s;...;%s\n", <sort-rules>, <sort-rules> ...
```

4638 The operands to the **order_start** keyword are optional. If present, the operands define rules to be
4639 applied when strings are compared. The number of operands define how many weights each
4640 element is assigned; if no operands are present, one **forward** operand is assumed. If present, the
4641 first operand defines rules to be applied when comparing strings using the first (primary)
4642 weight; the second when comparing strings using the second weight, and so on. Operands shall
4643 be separated by semicolons (';'). Each operand shall consist of one or more collation
4644 directives, separated by commas (','),. If the number of operands exceeds the
4645 {COLL_WEIGHTS_MAX} limit, the utility shall issue a warning message. The following
4646 directives shall be supported:

4647 **forward** Specifies that comparison operations for the weight level shall proceed from start
4648 of string towards the end of string.

4649 **backward** Specifies that comparison operations for the weight level shall proceed from end of
4650 string towards the beginning of string.

4651 **position** Specifies that comparison operations for the weight level shall consider the relative
4652 position of elements in the strings not subject to **IGNORE**. The string containing
4653 an element not subject to **IGNORE** after the fewest collating elements subject to
4654 **IGNORE** from the start of the compare collates first. If both strings contain a
4655 character not subject to **IGNORE** in the same relative position, the collating values
4656 assigned to the elements shall determine the ordering. In case of equality,
4657 subsequent characters not subject to **IGNORE** are considered in the same manner.

4658 The directives **forward** and **backward** are mutually-exclusive.

4659 For example:

```
4660 order_start forward;backward
```

4661 If no operands are specified, a single **forward** operand shall be assumed.

4662 The character (and collating element) order is defined by the order in which characters and
4663 elements are specified between the **order_start** and **order_end** keywords. This character order is
4664 used in range expressions in regular expressions (see Chapter 9). Weights assigned to the
4665 characters and elements define the collation sequence; in the absence of weights, the character
4666 order is also the collation sequence.

4667 The **position** keyword provides the capability to consider, in a compare, the relative position of
4668 characters not subject to **IGNORE**. As an example, consider the two strings "o-ring" and
4669 "or-ing". Assuming the hyphen is subject to **IGNORE** on the first pass, the two strings
4670 compare equal, and the position of the hyphen is immaterial. On second pass, all characters
4671 except the hyphen are subject to **IGNORE**, and in the normal case the two strings would again
4672 compare equal. By taking position into account, the first collates before the second.

4673 7.3.2.4 Collation Order

4674 The **order_start** keyword shall be followed by collating identifier entries. The syntax for the
4675 collating element entries is as follows:

```
4676 "%s %s;%s;...;%s\n", <collating-identifier>, <weight>, <weight>, ...
```

4677 Each *collating-identifier* shall consist of either a character (in any of the forms defined in Section
4678 7.3 (on page 145)), a *collating-element*, a *collating-symbol*, an ellipsis, or the special symbol
4679 **UNDEFINED**. The order in which collating elements are specified determines the character
4680 order sequence, such that each collating element shall compare less than the elements following
4681 it.

4682 A *collating-element* shall be used to specify multi-character collating elements, and indicates
4683 that the character sequence specified via the *collating-element* is to be collated as a unit and in
4684 the relative order specified by its place.

4685 A *collating-symbol* can be used to define a position in the relative order for use in weights. No
4686 weights can be specified with a *collating-symbol*.

4687 The ellipsis symbol specifies that a sequence of characters collates according to their encoded
4688 character values. It shall be interpreted as indicating that all characters with a coded character
4689 set value higher than the value of the character in the preceding line, and lower than the coded
4690 character set value for the character in the following line, in the current coded character set, shall
4691 be placed in the character collation order between the previous and the following character in
4692 ascending order according to their coded character set values. An initial ellipsis shall be
4693 interpreted as if the preceding line specified the NUL character, and a trailing ellipsis as if the
4694 following line specified the highest coded character set value in the current coded character set.
4695 An ellipsis shall be treated as invalid if the preceding or following lines do not specify characters
4696 in the current coded character set. The use of the ellipsis symbol ties the definition to a specific
4697 coded character set and may preclude the definition from being portable between
4698 implementations.

4699 The symbol **UNDEFINED** shall be interpreted as including all coded character set values not
4700 specified explicitly or via the ellipsis symbol. Such characters shall be inserted in the character
4701 collation order at the point indicated by the symbol, and in ascending order according to their
4702 coded character set values. If no **UNDEFINED** symbol is specified, and the current coded
4703 character set contains characters not specified in this section, the utility shall issue a warning
4704 message and place such characters at the end of the character collation order.

4705 The optional operands for each collation-element shall be used to define the primary, secondary,
4706 or subsequent weights for the collating element. The first operand specifies the relative primary

4707 weight, the second the relative secondary weight, and so on. Two or more collation-elements can
 4708 be assigned the same weight; they belong to the same *equivalence class* if they have the same
 4709 primary weight. Collation shall behave as if, for each weight level, elements subject to **IGNORE**
 4710 are removed, unless the **position** collation directive is specified for the corresponding level with
 4711 the **order_start** keyword. Then each successive pair of elements shall be compared according to
 4712 the relative weights for the elements. If the two strings compare equal, the process is repeated
 4713 for the next weight level, up to the limit {COLL_WEIGHTS_MAX}.

4714 Weights shall be expressed as characters (in any of the forms specified in Section 7.3 (on page
 4715 145)), *<collating-symbol>*s, *<collating-element>*s, an ellipsis, or the special symbol **IGNORE**. A
 4716 single character, a *<collating-symbol>*, or a *<collating-element>* shall represent the relative position
 4717 in the character collating sequence of the character or symbol, rather than the character or
 4718 characters themselves. Thus, rather than assigning absolute values to weights, a particular
 4719 weight is expressed using the relative order value assigned to a collating element based on its
 4720 order in the character collation sequence.

4721 One-to-many mapping is indicated by specifying two or more concatenated characters or
 4722 symbolic names. For example, if the character `<eszet>` is given the string "`<s><s>`" as a weight,
 4723 comparisons are performed as if all occurrences of the character `<eszet>` are replaced by
 4724 "`<s><s>`" (assuming that "`<s>`" has the collating weight "`<s>`"). If it is necessary to define
 4725 `<eszet>` and "`<s><s>`" as an equivalence class, then a collating element must be defined for the
 4726 string "ss".

4727 All characters specified via an ellipsis shall by default be assigned unique weights, equal to the
 4728 relative order of characters. Characters specified via an explicit or implicit **UNDEFINED** special
 4729 symbol shall by default be assigned the same primary weight (that is, they belong to the same
 4730 equivalence class). An ellipsis symbol as a weight shall be interpreted to mean that each
 4731 character in the sequence has unique weights, equal to the relative order of their character in the
 4732 character collation sequence. The use of the ellipsis as a weight shall be treated as an error if the
 4733 collating element is neither an ellipsis nor the special symbol **UNDEFINED**.

4734 The special keyword **IGNORE** as a weight shall indicate that when strings are compared using
 4735 the weights at the level where **IGNORE** is specified, the collating element shall be ignored; that
 4736 is, as if the string did not contain the collating element. In regular expressions and pattern
 4737 matching, all characters that are subject to **IGNORE** in their primary weight form an
 4738 equivalence class.

4739 An empty operand shall be interpreted as the collating element itself.

4740 For example, the order statement:

4741 `<a> <a> i <a>`

4742 is equal to:

4743 `<a>`

4744 An ellipsis can be used as an operand if the collating element was an ellipsis, and shall be
 4745 interpreted as the value of each character defined by the ellipsis.

4746 The collation order as defined in this section defines the interpretation of bracket expressions in
 4747 regular expressions (see Section 9.3.5 (on page 199)).

4748 For example:

```

4749     order_start  forward;backward
4750     UNDEFINED    IGNORE;IGNORE
4751     <LOW>
4752     <space>      <LOW>;<space>
4753     ...           <LOW>;...
4754     <a>           <a>;<a>
4755     <a-acute>    <a>;<a-acute>
4756     <a-grave>   <a>;<a-grave>
4757     <A>          <a>;<A>
4758     <A-acute>   <a>;<A-acute>
4759     <A-grave>   <a>;<A-grave>
4760     <ch>        <ch>;<ch>
4761     <Ch>        <ch>;<Ch>
4762     <s>          <s>;<s>
4763     <eszet>     "<s><s>";"<eszet><eszet>"
4764     order_end

```

4765 This example is interpreted as follows:

- 4766 1. The **UNDEFINED** means that all characters not specified in this definition (explicitly or
4767 via the ellipsis) shall be ignored for collation purposes; for regular expression purposes
4768 they are ordered first.
- 4769 2. All characters between <space> and 'a' shall have the same primary equivalence class
4770 and individual secondary weights based on their ordinal encoded values.
- 4771 3. All characters based on the uppercase or lowercase character 'a' belong to the same
4772 primary equivalence class.
- 4773 4. The multi-character collating element <ch> is represented by the collating symbol <ch>
4774 and belongs to the same primary equivalence class as the multi-character collating element
4775 <Ch>.

4776 7.3.2.5 The order_end Keyword

4777 The collating order entries shall be terminated with an **order_end** keyword.

4778 The collation sequence definition of the POSIX locale follows; the code listing depicts the
4779 *localedef* input.

```

4780 LC_COLLATE
4781 # This is the POSIX locale definition for the LC_COLLATE category.
4782 # The order is the same as in the ASCII codeset.
4783 order_start forward
4784 <NUL>
4785 <SOH>
4786 <STX>
4787 <ETX>
4788 <EOT>
4789 <ENQ>
4790 <ACK>
4791 <alert>
4792 <backspace>
4793 <tab>
4794 <newline>
4795 <vertical-tab>

```


4796	<form-feed>
4797	<carriage-return>
4798	<SO>
4799	<SI>
4800	<DLE>
4801	<DC1>
4802	<DC2>
4803	<DC3>
4804	<DC4>
4805	<NAK>
4806	<SYN>
4807	<ETB>
4808	<CAN>
4809	
4810	<SUB>
4811	<ESC>
4812	<IS4>
4813	<IS3>
4814	<IS2>
4815	<IS1>
4816	<space>
4817	<exclamation-mark>
4818	<quotation-mark>
4819	<number-sign>
4820	<dollar-sign>
4821	<percent-sign>
4822	<ampersand>
4823	<apostrophe>
4824	<left-parenthesis>
4825	<right-parenthesis>
4826	<asterisk>
4827	<plus-sign>
4828	<comma>
4829	<hyphen>
4830	<period>
4831	<slash>
4832	<zero>
4833	<one>
4834	<two>
4835	<three>
4836	<four>
4837	<five>
4838	<six>
4839	<seven>
4840	<eight>
4841	<nine>
4842	<colon>
4843	<semicolon>
4844	<less-than-sign>
4845	<equals-sign>
4846	<greater-than-sign>
4847	<question-mark>

4848	<commercial-at>
4849	<A>
4850	
4851	<C>
4852	<D>
4853	<E>
4854	<F>
4855	<G>
4856	<H>
4857	<I>
4858	<J>
4859	<K>
4860	<L>
4861	<M>
4862	<N>
4863	<O>
4864	<P>
4865	<Q>
4866	<R>
4867	<S>
4868	<T>
4869	<U>
4870	<V>
4871	<W>
4872	<X>
4873	<Y>
4874	<Z>
4875	<left-square-bracket>
4876	<backslash>
4877	<right-square-bracket>
4878	<circumflex>
4879	<underscore>
4880	<grave-accent>
4881	<a>
4882	
4883	<c>
4884	<d>
4885	<e>
4886	<f>
4887	<g>
4888	<h>
4889	<i>
4890	<j>
4891	<k>
4892	<l>
4893	<m>
4894	<n>
4895	<o>
4896	<p>
4897	<q>
4898	<r>
4899	<s>

```

4900     <t>
4901     <u>
4902     <v>
4903     <w>
4904     <x>
4905     <y>
4906     <z>
4907     <left-curly-bracket>
4908     <vertical-line>
4909     <right-curly-bracket>
4910     <tilde>
4911     <DEL>
4912     order_end
4913     #
4914     END LC_COLLATE

```

4915 7.3.3 LC_MONETARY

4916 The *LC_MONETARY* category shall define the rules and symbols that are used to format
 4917 XSI monetary numeric information. This information is available through the *localeconv()* function
 4918 and is used by the *strfmon()* function.

4919 XSI Some of the information is also available in an alternative form via the *nl_langinfo()* function
 4920 (see CRNCYSTR in <*langinfo.h*>).

4921 The following items are defined in this category of the locale. The item names are the keywords
 4922 recognized by the *localedef* utility when defining a locale. They are also similar to the member
 4923 names of the *lconv* structure defined in <*locale.h*>; see <*locale.h*> for the exact symbols in the
 4924 header. The *localeconv()* function returns {CHAR_MAX} for unspecified integer items and the
 4925 empty string (" ") for unspecified or size zero string items.

4926 In a locale definition file, the operands are strings, formatted as indicated by the grammar in
 4927 Section 7.4 (on page 176). For some keywords, the strings can contain only integers. Keywords
 4928 that are not provided, string values set to the empty string (" "), or integer keywords set to -1,
 4929 are used to indicate that the value is not available in the locale.

4930 **copy** Specify the name of an existing locale to be used as the definition of this
 4931 category. If this keyword is specified, no other keyword can be specified.

4932 **Note:** This is a *localedef* utility keyword, unavailable through
 4933 *localeconv()*.

4934 **int_curr_symbol** The international currency symbol. The operand is a four-character string,
 4935 with the first three characters containing the alphabetic international
 4936 currency symbol in accordance with those specified in the ISO 4217:1995
 4937 standard. The fourth character is the character used to separate the
 4938 international currency symbol from the monetary quantity.

4939 **currency_symbol** The string that shall be used as the local currency symbol.

4940 **mon_decimal_point** The operand is a string containing the symbol that shall be used as the
 4941 decimal delimiter (radix character) in monetary formatted quantities. In
 4942 contexts where standards (such as the ISO C standard) limit the
 4943 **mon_decimal_point** to a single byte, the result of specifying a multi-byte
 4944 operand is unspecified.

4945 **mon_thousands_sep** The operand is a string containing the symbol that shall be used as a
 4946 separator for groups of digits to the left of the decimal delimiter in
 4947 formatted monetary quantities. In contexts where standards limit the
 4948 **mon_thousands_sep** to a single byte, the result of specifying a multi-byte
 4949 operand is unspecified.

4950 **mon_grouping** Define the size of each group of digits in formatted monetary quantities.
 4951 The operand is a sequence of integers separated by semicolons. Each
 4952 integer specifies the number of digits in each group, with the initial
 4953 integer defining the size of the group immediately preceding the decimal
 4954 delimiter, and the following integers defining the preceding groups. If the
 4955 last integer is not -1, then the size of the previous group (if any) shall be
 4956 repeatedly used for the remainder of the digits. If the last integer is -1,
 4957 then no further grouping shall be performed.

4958 The following is an example of the interpretation of the **mon_grouping**
 4959 keyword. Assuming that the value to be formatted is 123456789 and the
 4960 **mon_thousands_sep** is ' ', then the following table shows the result.
 4961 The third column shows the equivalent string in the ISO C standard that
 4962 would be used by the *localeconv()* function to accommodate this
 4963 grouping.

mon_grouping	Formatted Value	ISO C String
3;-1	123456'789	"\3\177"
3	123'456'789	"\3"
3;2;-1	1234'56'789	"\3\2\177"
3;2	12'34'56'789	"\3\2"
-1	123456789	"\177"

4970 In these examples, the octal value of {CHAR_MAX} is 177.

4971 **positive_sign** A string that shall be used to indicate a non-negative-valued formatted
 4972 monetary quantity.

4973 **negative_sign** A string that shall be used to indicate a negative-valued formatted
 4974 monetary quantity.

4975 **int_frac_digits** An integer representing the number of fractional digits (those to the right
 4976 of the decimal delimiter) to be written in a formatted monetary quantity
 4977 using **int_curr_symbol**.

4978 **frac_digits** An integer representing the number of fractional digits (those to the right
 4979 of the decimal delimiter) to be written in a formatted monetary quantity
 4980 using **currency_symbol**.

4981 **p_cs_precedes** An integer set to 1 if the **currency_symbol** or **int_curr_symbol** precedes
 4982 the value for a monetary quantity with a non-negative value, and set to 0
 4983 if the symbol succeeds the value.

4984 **p_sep_by_space** An integer set to 0 if no space separates the **currency_symbol** or
 4985 **int_curr_symbol** from the value for a monetary quantity with a non-
 4986 negative value, set to 1 if a space separates the symbol from the value,
 4987 and set to 2 if a space separates the symbol and the sign string, if adjacent.

4988 **n_cs_precedes** An integer set to 1 if the **currency_symbol** or **int_curr_symbol** precedes
 4989 the value for a monetary quantity with a negative value, and set to 0 if the
 4990 symbol succeeds the value.

- 4991 **n_sep_by_space** An integer set to 0 if no space separates the **currency_symbol** or
- 4992 **int_curr_symbol** from the value for a monetary quantity with a negative
- 4993 value, set to 1 if a space separates the symbol from the value, and set to 2
- 4994 if a space separates the symbol and the sign string, if adjacent.

- 4995 **p_sign_posn** An integer set to a value indicating the positioning of the **positive_sign**
- 4996 for a monetary quantity with a non-negative value. The following integer
- 4997 values shall be recognized for both **p_sign_posn** and **n_sign_posn**:

- 4998 0 Parentheses enclose the quantity and the **currency_symbol** or
- 4999 **int_curr_symbol**.

- 5000 1 The sign string precedes the quantity and the **currency_symbol** or
- 5001 **int_curr_symbol**.

- 5002 2 The sign string succeeds the quantity and the **currency_symbol** or
- 5003 **int_curr_symbol**.

- 5004 3 The sign string precedes the **currency_symbol** or **int_curr_symbol**.
- 5005 4 The sign string succeeds the **currency_symbol** or **int_curr_symbol**.

- 5006 **n_sign_posn** An integer set to a value indicating the positioning of the **negative_sign**
- 5007 for a negative formatted monetary quantity.

5008 The following table shows the result of various combinations:

		p_sep_by_space		
		2	1	0
p_cs_precedes = 1	p_sign_posn = 0	(\$1.25)	(\$ 1.25)	(\$1.25)
	p_sign_posn = 1	+ \$1.25	+\$ 1.25	+\$1.25
	p_sign_posn = 2	\$1.25 +	\$ 1.25+	\$1.25+
	p_sign_posn = 3	+ \$1.25	+\$ 1.25	+\$1.25
	p_sign_posn = 4	\$ +1.25	\$+ 1.25	\$+1.25
p_cs_precedes = 0	p_sign_posn = 0	(1.25 \$)	(1.25 \$)	(1.25\$)
	p_sign_posn = 1	+1.25 \$	+1.25 \$	+1.25\$
	p_sign_posn = 2	1.25\$ +	1.25 \$+	1.25\$+
	p_sign_posn = 3	1.25+ \$	1.25 +\$	1.25+\$
	p_sign_posn = 4	1.25\$ +	1.25 \$+	1.25\$+

5021 The monetary formatting definitions for the POSIX locale follow; the code listing depicting the

5022 XSI *localedef* input, the table representing the same information with the addition of *localeconv()* and

5023 *nl_langinfo()* formats. All values are unspecified in the POSIX locale.

```

5024 LC_MONETARY
5025 # This is the POSIX locale definition for
5026 # the LC_MONETARY category.
5027 #
5028 int_curr_symbol      " "
5029 currency_symbol     " "
5030 mon_decimal_point   " "
5031 mon_thousands_sep  " "
5032 mon_grouping        -1
5033 positive_sign       " "
5034 negative_sign       " "
5035 int_frac_digits     -1
5036 frac_digits         -1
    
```

```

5037 p_cs_precedes -1
5038 p_sep_by_space -1
5039 n_cs_precedes -1
5040 n_sep_by_space -1
5041 p_sign_posn -1
5042 n_sign_posn -1
5043 #
5044 END LC_MONETARY

```

	Item	POSIX locale Value	langinfo Constant	localeconv() Value	localedef Value
5045	currency_symbol	N/A	CRNCYSTR	" "	" "
5046	frac_digits	N/A	—	CHAR_MAX	-1
5047	int_curr_symbol	N/A	—	" "	" "
5048	int_frac_digits	N/A	—	CHAR_MAX	-1
5049	mon_decimal_point	N/A	—	" "	" "
5050	mon_thousands_sep	N/A	—	" "	" "
5051	mon_grouping	N/A	—	" "	" "
5052	positive_sign	N/A	—	" "	" "
5053	negative_sign	N/A	—	" "	" "
5054	p_cs_precedes	N/A	CRNCYSTR	CHAR_MAX	-1
5055	n_cs_precedes	N/A	CRNCYSTR	CHAR_MAX	-1
5056	p_sep_by_space	N/A	—	CHAR_MAX	-1
5057	n_sep_by_space	N/A	—	CHAR_MAX	-1
5058	p_sign_posn	N/A	—	CHAR_MAX	-1
5059	n_sign_posn	N/A	—	CHAR_MAX	-1

5062 XSI In the preceding table, the **langinfo Constant** column represents an XSI-conformant extension.
5063 The entry N/A indicates that the value is not available in the POSIX locale.

5064 7.3.4 LC_NUMERIC

5065 The *LC_NUMERIC* category shall define the rules and symbols that are used to format non-
5066 XSI monetary numeric information. This information is available through the *localeconv()* function.
5067 Some of the information is also available in an alternative form via the *nl_langinfo()* function.

5068 The following items are defined in this category of the locale. The item names are the keywords
5069 recognized by the *localedef* utility when defining a locale. They are also similar to the member
5070 names of the *lconv* structure defined in *<locale.h>*; see *<locale.h>* for the exact symbols in the
5071 header. The *localeconv()* function returns {CHAR_MAX} for unspecified integer items and the
5072 empty string (" ") for unspecified or size zero string items.

5073 In a locale definition file, the operands are strings, formatted as indicated by the grammar in
5074 Section 7.4 (on page 176). For some keywords, the strings can only contain integers. Keywords
5075 that are not provided, string values set to the empty string (" "), or integer keywords set to -1,
5076 shall be used to indicate that the value is not available in the locale. The following keywords
5077 shall be recognized:

5078 **copy** Specify the name of an existing locale to be used as the definition of this
5079 category. If this keyword is specified, no other keyword can be specified.

5080 **Note:** This is a *localedef* utility keyword, unavailable through *localeconv()*.

5081 **decimal_point** The operand is a string containing the symbol that shall be used as the
5082 decimal delimiter (radix character) in numeric, non-monetary formatted
5083 quantities. This keyword cannot be omitted and cannot be set to the empty

5084 string. In contexts where standards limit the **decimal_point** to a single byte,
 5085 the result of specifying a multi-byte operand shall be unspecified.

5086 **thousands_sep** The operand is a string containing the symbol that shall be used as a separator
 5087 for groups of digits to the left of the decimal delimiter in numeric, non-
 5088 monetary formatted monetary quantities. In contexts where standards limit
 5089 the **thousands_sep** to a single byte, the result of specifying a multi-byte
 5090 operand shall be unspecified.

5091 **grouping** Define the size of each group of digits in formatted non-monetary quantities.
 5092 The operand is a sequence of integers separated by semicolons. Each integer
 5093 specifies the number of digits in each group, with the initial integer defining
 5094 the size of the group immediately preceding the decimal delimiter, and the
 5095 following integers defining the preceding groups. If the last integer is not -1,
 5096 then the size of the previous group (if any) shall be repeatedly used for the
 5097 remainder of the digits. If the last integer is -1, then no further grouping shall
 5098 be performed.

5099 The non-monetary numeric formatting definitions for the POSIX locale follow; the code listing
 5100 depicting the *localedef* input, the table representing the same information with the addition of
 5101 XSI *localeconv()* values, and *nl_langinfo()* constants.

```
5102 LC_NUMERIC
5103 # This is the POSIX locale definition for
5104 # the LC_NUMERIC category.
5105 #
5106 decimal_point      "<period>"
5107 thousands_sep      " "
5108 grouping           -1
5109 #
5110 END LC_NUMERIC
```

Item	POSIX Locale Value	langinfo Constant	localeconv() Value	localedef Value
decimal_point	"."	RADIXCHAR	"."	."
thousands_sep	N/A	THOUSEP	" "	" "
grouping	N/A	—	" -1"	-1

5116 **Notes to Reviewers**

5117 *This section with side shading will not appear in the final copy. - Ed.*

5118 D1, XBD, ERN 112 asked why the grouping in the POSIX locale is -1, but the grouping line in the
 5119 POSIX Locale Value column of this table is N/A. The response from Gary Miller (via Mark
 5120 Brown) was that they are saying the same thing; the -1 means that there is no grouping, therefore
 5121 the grouping is not applicable.

5122 XSI In the preceding table, the **langinfo Constant** column represents an XSI-conforming extension.
 5123 The entry N/A indicates that the value is not available in the POSIX locale.

5124 **7.3.5 LC_TIME**

5125 The *LC_TIME* category shall define the interpretation of the field descriptors supported by the
 5126 XSI *date* utility and affects the behavior of the *strptime()*, *wcsftime()*, *strptime()*, and *nl_langinfo()*
 5127 functions. Because the interfaces for C-language access and locale definition differ significantly,
 5128 they are described separately.

5129 **7.3.5.1 LC_TIME Locale Definition**

5130 For locale definition, the following mandatory keywords shall be recognized:

5131 **copy** Specify the name of an existing locale to be used as the definition of this
 5132 category. If this keyword is specified, no other keyword can be specified.

5133 **abday** Define the abbreviated weekday names, corresponding to the *%a* field
 5134 descriptor (conversion specification in the *strptime()*, *wcsftime()*, and *strptime()*
 5135 functions). The operand consists of seven semicolon-separated strings, each
 5136 surrounded by double-quotes. The first string shall be the abbreviated name of
 5137 the day corresponding to Sunday, the second the abbreviated name of the day
 5138 corresponding to Monday, and so on.

5139 **day** Define the full weekday names, corresponding to the *%A* field descriptor. The
 5140 operand consists of seven semicolon-separated strings, each surrounded by
 5141 double-quotes. The first string is the full name of the day corresponding to
 5142 Sunday, the second the full name of the day corresponding to Monday, and so
 5143 on.

5144 **abmon** Define the abbreviated month names, corresponding to the *%b* field
 5145 descriptor. The operand consists of twelve semicolon-separated strings, each
 5146 surrounded by double-quotes. The first string shall be the abbreviated name of
 5147 the first month of the year (January), the second the abbreviated name of the
 5148 second month, and so on.

5149 **mon** Define the full month names, corresponding to the *%B* field descriptor. The
 5150 operand consists of twelve semicolon-separated strings, each surrounded by
 5151 double-quotes. The first string shall be the full name of the first month of the
 5152 year (January), the second the full name of the second month, and so on.

5153 **d_t_fmt** Define the appropriate date and time representation, corresponding to the *%c*
 5154 field descriptor. The operand consists of a string, and can contain any
 5155 combination of characters and field descriptors. In addition, the string can
 5156 contain escape sequences defined in the table in Table 5-1 (on page 130) ('\\',
 5157 '\\a', '\\b', '\\f', '\\n', '\\r', '\\t', '\\v').

5158 **d_fmt** Define the appropriate date representation, corresponding to the *%x* field
 5159 descriptor. The operand consists of a string, and can contain any combination
 5160 of characters and field descriptors. In addition, the string can contain escape
 5161 sequences defined in the table in Table 5-1 (on page 130).

5162 **t_fmt** Define the appropriate time representation, corresponding to the *%X* field
 5163 descriptor. The operand consists of a string, and can contain any combination
 5164 of characters and field descriptors. In addition, the string can contain escape
 5165 sequences defined in the table in Table 5-1 (on page 130).

5166 **am_pm** Define the appropriate representation of the *ante meridiem* and *post meridiem*
 5167 strings, corresponding to the *%p* field descriptor. The operand consists of two
 5168 strings, separated by a semicolon, each surrounded by double-quotes. The
 5169 first string shall represent the *ante meridiem* designation, the last string the *post*

5170		<i>meridiem</i> designation.
5171	t_fmt_ampm	Define the appropriate time representation in the 12-hour clock format with am_pm , corresponding to the %r field descriptor. The operand consists of a string and can contain any combination of characters and field descriptors. If the string is empty, the 12-hour format is not supported in the locale.
5172		
5173		
5174		
5175	era	Define how years are counted and displayed for each era in a locale. The operand consists of semicolon-separated strings. Each string is an era description segment with the format:
5176		
5177		
5178		<i>direction:offset:start_date:end_date:era_name:era_format</i>
5179		according to the definitions below. There can be as many era description segments as are necessary to describe the different eras.
5180		
5181	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.
5182		
5183		
5184	<i>direction</i>	Either a '+' or a '-' character. The '+' character indicates that years closer to the <i>start_date</i> have lower numbers than those closer to the <i>end_date</i> . The '-' character indicates that years closer to the <i>start_date</i> have higher numbers than those closer to the <i>end_date</i> .
5185		
5186		
5187		
5188		
5189	<i>offset</i>	The number of the year closest to the <i>start_date</i> in the era, corresponding to the %Ey field descriptor.
5190		
5191	<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 are represented as negative numbers.
5192		
5193		
5194	<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 "-*" indicates that the ending date is the beginning of time. The value "+*" indicates that the ending date is the end of time.
5195		
5196		
5197		
5198	<i>era_name</i>	A string representing the name of the era, corresponding to the %EC field descriptor.
5199		
5200	<i>era_format</i>	A string for formatting the year in the era, corresponding to the %EY field descriptor.
5201		
5202	era_d_fmt	Define the format of the date in alternative era notation, corresponding to the %Ex field descriptor.
5203		
5204	era_t_fmt	Define the locale's appropriate alternative time format, corresponding to the %EX field descriptor.
5205		
5206	era_d_t_fmt	Define the locale's appropriate alternative date and time format, corresponding to the %Ec field descriptor.
5207		
5208	alt_digits	Define alternative symbols for digits, corresponding to the %O field descriptor modifier. The operand consists of semicolon-separated strings, each surrounded by double-quotes. The first string is the alternative symbol corresponding with zero, the second string the symbol corresponding with one, and so on. Up to 100 alternative symbol strings can be specified. The %O modifier indicates that the string corresponding to the value specified via the field descriptor is used instead of the value.
5209		
5210		
5211		
5212		
5213		
5214		

5215 7.3.5.2 LC_TIME C-Language Access

5216 XSI The following information can be accessed. These correspond to constants defined in
5217 <langinfo.h> and used as arguments to the *nl_langinfo()* function.

5218	ABDAY_x	The abbreviated weekday names (for example Sun), where <i>x</i> is a number from 1 to 7.
5219		
5220	DAY_x	The full weekday names (for example Sunday), where <i>x</i> is a number from 1 to 7.
5221		
5222	ABMON_x	The abbreviated month names (for example Jan), where <i>x</i> is a number from 1 to 12.
5223		
5224	MON_x	The full month names (for example January), where <i>x</i> is a number from 1 to 12.
5225		
5226	D_T_FMT	The appropriate date and time representation.
5227	D_FMT	The appropriate date representation.
5228	T_FMT	The appropriate time representation.
5229	AM_STR	The appropriate ante-meridiem affix.
5230	PM_STR	The appropriate post-meridiem affix.
5231	T_FMT_AMPM	The appropriate time representation in the 12-hour clock format with AM_STR and PM_STR.
5232		
5233	ERA	The era description segments, which describe how years are counted and displayed for each era in a locale. Each era description segment has the format:
5234		
5235		<i>direction:offset:start_date:end_date:era_name:era_format</i>
5236		according to the definitions below. There are as many era description segments as are necessary to describe the different eras. Era description segments are separated by semicolons.
5237		
5238		
5239		<i>direction</i> Either a '+' or a '-' character. The '+' character indicates that years closer to the <i>start_date</i> have lower numbers than those closer to the <i>end_date</i> . The '-' character indicates that years closer to the <i>start_date</i> have higher numbers than those closer to the <i>end_date</i> .
5240		
5241		
5242		
5243		
5244		<i>offset</i> The number of the year closest to the <i>start_date</i> in the era.
5245		
5246		<i>start_date</i> A date in the form <i>yyyy/mm/dd</i> , where <i>yyyy</i> , <i>mm</i> , and <i>dd</i> are the year, month, and day numbers respectively of the start of the era. Years prior to AD 1 are represented as negative numbers.
5247		
5248		<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 "-*" indicates that the ending date is the beginning of time. The value "+*" indicates that the ending date is the end of time.
5249		
5250		
5251		
5252		<i>era_name</i> The era, corresponding to the %EC conversion specification.
5253		
5254		<i>era_format</i> The format of the year in the era, corresponding to the %EY conversion specification.
5255	ERA_D_FMT	The era date format.

- 5256 ERA_T_FMT The locale's appropriate alternative time format, corresponding to the %EX
- 5257 field descriptor.
- 5258 ERA_D_T_FMT The locale's appropriate alternative date and time format, corresponding to
- 5259 the %Ec field descriptor.
- 5260 ALT_DIGITS The alternative symbols for digits, corresponding to the %O conversion
- 5261 specification modifier. The value consists of semicolon-separated symbols.
- 5262 The first is the alternative symbol corresponding to zero, the second is the
- 5263 symbol corresponding to one, and so on. Up to 100 alternative symbols may
- 5264 be specified.

5265 The following table displays the correspondence between the items described above and the
 5266 conversion specifiers used by the *date* utility and the *strftime()*, *wcsftime()*, and *strptime()*
 5267 functions.

5268

5269

localedef Keyword	langinfo Constant	Conversion Specifier
5270 abday	ABDAY_x	%a
5271 day	DAY_x	%A
5272 abmon	ABMON_x	%b
5273 mon	MON	%B
5274 d_t_fmt	D_T_FMT	%c
5275 d_fmt	D_FMT	%x
5276 t_fmt	T_FMT	%X
5277 am_pm	AM_STR	%p
5278 am_pm	PM_STR	%p
5279 t_fmt_ampm	T_FMT_AMP	%r
5280 era	ERA	%EC, %Ey, %EY
5281 era_d_fmt	ERA_D_FMT	%Ex
5282 era_t_fmt	ERA_T_FMT	%EX
5283 era_d_t_fmt	ERA_D_T_FMT	%Ec
5284 alt_digits	ALT_DIGITS	%O

5285 In the preceding table, the **langinfo Constant** column represents an XSI-conformant extension.

5286 7.3.5.3 LC_TIME General Information

5287 The following is an example for Japan that supports the current plus last three Emperors and
 5288 reverts to Western style numbering for years prior to the Meiji era. The example also allows for
 5289 the custom of using a special name for the first year of an era instead of using 1. (The examples
 5290 substitute romaji where kanji should be used.)

```

5291     era_d_fmt "%EY%mgatsu%dnichi (%a)"
5292     era      "+:2:1990/01/01:+*:Heisei:%EC%Eyten";\
5293             "+:1:1989/01/08:1989/12/31:Heisei:%ECgannen";\
5294             "+:2:1927/01/01:1989/01/07:Shouwa:%EC%Eyten";\
5295             "+:1:1926/12/25:1926/12/31:Shouwa:%ECgannen";\
5296             "+:2:1913/01/01:1926/12/24:Taishou:%EC%Eyten";\
5297             "+:1:1912/07/30:1912/12/31:Taishou:%ECgannen";\
5298             "+:2:1869/01/01:1912/07/29:Meiji:%EC%Eyten";\
5299             "+:1:1868/09/08:1868/12/31:Meiji:%ECgannen";\
5300             "-:1868:1868/09/07:-*::%Ey"
```

5301 Assuming that the current date is September 21, 1991, a request to *date* or *strftime()* would yield
5302 the following results:

```
5303      %Ec - Heisei3nen9gatsu21nichi (Sat) 14:39:26
5304      %EC - Heisei
5305      %Ex - Heisei3nen9gatsu21nichi (Sat)
5306      %Ey - 3
5307      %EY - Heisei3nen
```

5308 Example era definitions for the Republic of China:

```
5309      era      "+:2:1913/01/01:+*:ChungHwaMingGuo:%EC%EyNen";\
5310              "+:1:1912/1/1:1912/12/31:ChungHwaMingGuo:%ECYuenNen";\
5311              "+:1:1911/12/31:-*:MingChien:%EC%EyNen"
```

5312 Example definitions for the Christian Era:

```
5313      era      "+:0:0000/01/01:+*:AD:%EC %Ey";\
5314              "+:1:-0001/12/31:-*:BC:%Ey %EC"
```

5315 The *LC_TIME* category definition of the POSIX locale follows; the code listing depicts the
5316 XSI *localedef* input; the table depicts the *langinfo* items defined in this category.

```
5317 LC_TIME
5318 # This is the POSIX locale definition for
5319 # the LC_TIME category.
5320 #
5321 # Abbreviated weekday names (%a)
5322 abday      "<S><u><n>"; "<M><o><n>"; "<T><u><e>"; "<W><e><d>"; \
5323            "<T><h><u>"; "<F><r><i>"; "<S><a><t>"
5324 #
5325 # Full weekday names (%A)
5326 day        "<S><u><n><d><a><y>"; "<M><o><n><d><a><y>"; \
5327            "<T><u><e><s><d><a><y>"; "<W><e><d><n><e><s><d><a><y>"; \
5328            "<T><h><u><r><s><d><a><y>"; "<F><r><i><d><a><y>"; \
5329            "<S><a><t><u><r><d><a><y>"
5330 #
5331 # Abbreviated month names (%b)
5332 abmon      "<J><a><n>"; "<F><e><b>"; "<M><a><r>"; \
5333            "<A><p><r>"; "<M><a><y>"; "<J><u><n>"; \
5334            "<J><u><l>"; "<A><u><g>"; "<S><e><p>"; \
5335            "<O><c><t>"; "<N><o><v>"; "<D><e><c>"
5336 #
5337 # Full month names (%B)
5338 mon        "<J><a><n><u><a><r><y>"; "<F><e><b><r><u><a><r><y>"; \
5339            "<M><a><r><c><h>"; "<A><p><r><i><l>"; \
5340            "<M><a><y>"; "<J><u><n><e>"; \
5341            "<J><u><l><y>"; "<A><u><g><u><s><t>"; \
5342            "<S><e><p><t><e><m><b><e><r>"; "<O><c><t><o><b><e><r>"; \
5343            "<N><o><v><e><m><b><e><r>"; "<D><e><c><e><m><b><e><r>"
5344 #
5345 # Equivalent of AM/PM (%p)      "AM"; "PM"
5346 am_pm      "<A><M>"; "<P><M>"
5347 #
5348 # Appropriate date and time representation (%c)
5349 #      "%a %b %e %H:%M:%S %Y"
```

```

5350     d_t_fmt      "<percent-sign><a><space><percent-sign><b>\
5351                 <space><percent-sign><e><space><percent-sign><H>\
5352                 <colon><percent-sign><M><colon><percent-sign><S>\
5353                 <space><percent-sign><Y>"
5354     #
5355     # Appropriate date representation (%x)    "%m/%d/%y"
5356     d_fmt        "<percent-sign><m><slash><percent-sign><d>\
5357                 <slash><percent-sign><y>"
5358     #
5359     # Appropriate time representation (%X)    "%H:%M:%S"
5360     t_fmt        "<percent-sign><H><colon><percent-sign><M>\
5361                 <colon><percent-sign><S>"
5362     #
5363     # Appropriate 12-hour time representation (%r) "%I:%M:%S %p"
5364     t_fmt_ampm   "<percent-sign><I><colon><percent-sign><M><colon>\
5365                 <percent-sign><S><space><percent_sign><p>"
5366     #
5367     END LC_TIME

```

5368
5369
5370 XSI
5371
5372
5373
5374
5375
5376
5377
5378
5379
5380
5381
5382
5383
5384
5385
5386
5387
5388
5389
5390
5391

Item	POSIX Locale Value	Item	POSIX Locale Value
D_T_FMT	"%a %b %e %H:%M:%S %Y"	MON_3	"March"
D_FMT	"%m/%d/%y"	MON_4	"April"
T_FMT	"%H:%M:%S"	MON_5	"May"
AM_STR	"AM"	MON_6	"June"
PM_STR	"PM"	MON_7	"July"
T_FMT_AMP	"%I:%M:%S %p"	MON_8	"August"
DAY_1	"Sunday"	MON_9	"September"
DAY_2	"Monday"	MON_10	"October"
DAY_3	"Tuesday"	MON_11	"November"
DAY_4	"Wednesday"	MON_12	"December"
DAY_5	"Thursday"	ABMON_1	"Jan"
DAY_6	"Friday"	ABMON_2	"Feb"
DAY_7	"Saturday"	ABMON_3	"Mar"
ABDAY_1	"Sun"	ABMON_4	"Apr"
ABDAY_2	"Mon"	ABMON_5	"May"
ABDAY_3	"Tue"	ABMON_6	"Jun"
ABDAY_4	"Wed"	ABMON_7	"Jul"
ABDAY_5	"Thu"	ABMON_8	"Aug"
ABDAY_6	"Fri"	ABMON_9	"Sep"
ABDAY_7	"Sat"	ABMON_10	"Oct"
MON_1	"January"	ABMON_11	"Nov"
MON_2	"February"	ABMON_12	"Dec"

5392 **7.3.6 LC_MESSAGES**

5393 The *LC_MESSAGES* category shall define the format and values for affirmative and negative
5394 responses.

5395 XSI The message catalog used by the standard utilities and selected by the *catopen()* function shall be
5396 determined by the setting of *NLSPATH*; see Chapter 8 (on page 187). The *LC_MESSAGES*
5397 category can be specified as part of an *NLSPATH* substitution field.

5398 XSI The following keywords shall be recognized as part of the locale definition file. The
5399 *nl_langinfo()* function accepts uppercase versions of the first four keywords.

5400 **copy** Specify the name of an existing locale to be used as the definition of this category.
5401 If this keyword is specified, no other keyword can be specified.

5402 **yesexpr** The operand consists of an extended regular expression (see Section 9.4 (on page
5403 203)) that describes the acceptable affirmative response to a question expecting an
5404 affirmative or negative response.

5405 **noexpr** The operand consists of an extended regular expression that describes the
5406 acceptable negative response to a question expecting an affirmative or negative
5407 response.

5408 The format and values for affirmative and negative responses of the POSIX locale follow; the
5409 code listing depicting the *localedef* input, the table representing the same information with the
5410 XSI addition of *nl_langinfo()* constants.

```
5411 LC_MESSAGES
5412 # This is the POSIX locale definition for
5413 # the LC_MESSAGES category.
5414 #
```

```

5415     yesexpr "<circumflex><left-square-bracket><y><Y><right-square-bracket>"
5416     #
5417     noexpr  "<circumflex><left-square-bracket><n><N><right-square-bracket>"
5418     #
5419 XSI   yesstr  "yes"
5420     nostr   "no"
5421     END LC_MESSAGES

```

	localedef Keyword	langinfo Constant	POSIX Locale Value
5422	yesexpr	YESEXPR	"^[yY]"
5423	noexpr	NOEXPR	"^[nN]"
5424			
5425	XSI yesstr	YESSTR	"yes" (LEGACY)
5426	XSI nostr	NOSTR	"no" (LEGACY)

5427 **7.3.6.1** *LC_MESSAGES Application Usage*

5428 XSI The **yesstr** and **nostr** locale keywords and the YESSTR and NOSTR *langinfo* items were formerly
5429 used to match user affirmative and negative responses. In IEEE Std. 1003.1-200x, the **yesexpr**,
5430 **noexpr**, YESEXPR, and NOEXPR extended regular expressions have replaced them. However,
5431 they have been retained for backward compatibility to allow an application to include a sample
5432 desired response in a prompting message. They are marked LEGACY. Applications should use
5433 the general locale-based messaging facilities to issue such prompting messages.

5434 7.4 Locale Definition Grammar

5435 The grammar and lexical conventions in this section shall together describe the syntax for the
 5436 locale definition source. The general conventions for this style of grammar are described in the
 5437 Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 1.10, Grammar Conventions. The
 5438 grammar shall take precedence over the text in this chapter.

5439 7.4.1 Locale Lexical Conventions

5440 The lexical conventions for the locale definition grammar are described in this section.

5441 The following tokens shall be processed (in addition to those string constants shown in the
 5442 grammar):

5443	LOC_NAME	A string of characters representing the name of a locale.
5444	CHAR	Any single character.
5445	NUMBER	A decimal number, represented by one or more decimal digits.
5446	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.
5447		
5448		
5449	COLLELEMENT	A symbolic name, enclosed between angle brackets, which cannot duplicate either any charmap symbol or a COLLSYMBOL symbol.
5450		
5451	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.
5452		
5453		
5454		
5455	CHARSYMBOL	A symbolic name, enclosed between angle brackets, from the current charmap (if any).
5456		
5457	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.
5458		
5459		
5460		
5461	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.
5462		
5463		
5464		
5465	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.
5466		
5467		
5468		
5469	ELLIPSIS	The string " . . . ".
5470	EXTENDED_REG_EXP	An extended regular expression as defined in the grammar in Section 9.5 (on page 206).
5471		
5472	EOL	The line termination character newline.

5473 **7.4.2 Locale Grammar**

5474 This section presents the grammar for the locale definition.

```

5475 %token          LOC_NAME
5476 %token          CHAR
5477 %token          NUMBER
5478 %token          COLLSYMBOL COLLELEMENT
5479 %token          CHARSYMBOL OCTAL_CHAR HEX_CHAR DECIMAL_CHAR
5480 %token          ELLIPSIS
5481 %token          EXTENDED_REG_EXP
5482 %token          EOL
5483 %start          locale_definition
5484 %%
5485 locale_definition : global_statements locale_categories
5486                  | locale_categories
5487                  ;
5488 global_statements : global_statements symbol_redefine
5489                  | symbol_redefine
5490                  ;
5491 symbol_redefine   : 'escape_char' CHAR EOL
5492                  | 'comment_char' CHAR EOL
5493                  ;
5494 locale_categories : locale_categories locale_category
5495                  | locale_category
5496                  ;
5497 locale_category  : lc_ctype | lc_collate | lc_messages
5498                  | lc_monetary | lc_numeric | lc_time
5499                  ;
5500 /* The following grammar rules are common to all categories */
5501 char_list        : char_list char_symbol
5502                  | char_symbol
5503                  ;
5504 char_symbol      : CHAR | CHARSYMBOL
5505                  | OCTAL_CHAR | HEX_CHAR | DECIMAL_CHAR
5506                  ;
5507 elem_list        : elem_list char_symbol
5508                  | elem_list COLLSYMBOL
5509                  | elem_list COLLELEMENT
5510                  | char_symbol
5511                  | COLLSYMBOL
5512                  | COLLELEMENT
5513                  ;
5514 symb_list        : symb_list COLLSYMBOL
5515                  | COLLSYMBOL
5516                  ;

```

```

5517     locale_name      : LOC_NAME
5518     | ''' LOC_NAME '''
5519     ;

5520     /* The following is the LC_CTYPE category grammar */

5521     lc_ctype          : ctype_hdr ctype_keywords      ctype_tlr
5522     | ctype_hdr 'copy' locale_name EOL ctype_tlr
5523     ;

5524     ctype_hdr         : 'LC_CTYPE' EOL
5525     ;

5526     ctype_keywords   : ctype_keywords ctype_keyword
5527     | ctype_keyword
5528     ;

5529     ctype_keyword    : charclass_keyword charclass_list EOL
5530     | charconv_keyword charconv_list EOL
5531     | 'charclass' charclass_namelist EOL
5532     ;

5533     charclass_namelist : charclass_namelist ';' CHARCLASS
5534     | CHARCLASS
5535     ;

5536     charclass_keyword : 'upper' | 'lower' | 'alpha' | 'digit'
5537     | 'punct' | 'xdigit' | 'space' | 'print'
5538     | 'graph' | 'blank' | 'cntrl' | 'alnum'
5539     | CHARCLASS
5540     ;

5541     charclass_list    : charclass_list ';' char_symbol
5542     | charclass_list ';' ELLIPSIS ';' char_symbol
5543     | char_symbol
5544     ;

5545     charconv_keyword  : 'toupper'
5546     | 'tolower'
5547     ;

5548     charconv_list     : charconv_list ';' charconv_entry
5549     | charconv_entry
5550     ;

5551     charconv_entry    : '(' char_symbol ',' char_symbol ')'
5552     ;

5553     ctype_tlr        : 'END' 'LC_CTYPE' EOL
5554     ;

5555     /* The following is the LC_COLLATE category grammar */

5556     lc_collate        : collate_hdr collate_keywords      collate_tlr
5557     | collate_hdr 'copy' locale_name EOL collate_tlr
5558     ;

5559     collate_hdr       : 'LC_COLLATE' EOL
5560     ;

```

```

5561     collate_keywords      :           order_statements
5562                           | opt_statements order_statements
5563                           ;

5564     opt_statements         : opt_statements collating_symbols
5565                           | opt_statements collating_elements
5566                           | collating_symbols
5567                           | collating_elements
5568                           ;

5569     collating_symbols      : 'collating-symbol' COLLSYMBOL EOL
5570                           ;

5571     collating_elements     : 'collating-element' COLLELEMENT
5572                           | 'from' ''' elem_list ''' EOL
5573                           ;

5574     order_statements       : order_start collation_order order_end
5575                           ;

5576     order_start            : 'order_start' EOL
5577                           | 'order_start' order_opts EOL
5578                           ;

5579     order_opts              : order_opts ';' order_opt
5580                           | order_opt
5581                           ;

5582     order_opt              : order_opt ',' opt_word
5583                           | opt_word
5584                           ;

5585     opt_word                : 'forward' | 'backward' | 'position'
5586                           ;

5587     collation_order        : collation_order collation_entry
5588                           | collation_entry
5589                           ;

5590     collation_entry        : COLLSYMBOL EOL
5591                           | collation_element weight_list EOL
5592                           | collation_element                EOL
5593                           ;

5594     collation_element       : char_symbol
5595                           | COLLELEMENT
5596                           | ELLIPSIS
5597                           | 'UNDEFINED'
5598                           ;

5599     weight_list            : weight_list ';' weight_symbol
5600                           | weight_list ';'
5601                           | weight_symbol
5602                           ;

5603     weight_symbol          : /* empty */
5604                           | char_symbol
5605                           | COLLSYMBOL
5606                           | ''' elem_list '''

```

```

5607         | ''' symb_list '''
5608         | ELLIPSIS
5609         | 'IGNORE'
5610         ;
5611 order_end      : 'order_end' EOL
5612         ;
5613 collate_tlr    : 'END' 'LC_COLLATE' EOL
5614         ;
5615 /* The following is the LC_MESSAGES category grammar */
5616 lc_messages    : messages_hdr messages_keywords      messages_tlr
5617         | messages_hdr 'copy' locale_name EOL messages_tlr
5618         ;
5619 messages_hdr   : 'LC_MESSAGES' EOL
5620         ;
5621 messages_keywords : messages_keywords messages_keyword
5622         | messages_keyword
5623         ;
5624 messages_keyword : 'yesexpr' ''' EXTENDED_REG_EXP ''' EOL
5625         | 'noexpr' ''' EXTENDED_REG_EXP ''' EOL
5626         | 'yesstr' ''' char_list ''' EOL
5627         | 'nostr' ''' char_list ''' EOL
5628         ;
5629 messages_tlr   : 'END' 'LC_MESSAGES' EOL
5630         ;
5631 /* The following is the LC_MONETARY category grammar */
5632 lc_monetary    : monetary_hdr monetary_keywords      monetary_tlr
5633         | monetary_hdr 'copy' locale_name EOL monetary_tlr
5634         ;
5635 monetary_hdr   : 'LC_MONETARY' EOL
5636         ;
5637 monetary_keywords : monetary_keywords monetary_keyword
5638         | monetary_keyword
5639         ;
5640 monetary_keyword : mon_keyword_string mon_string EOL
5641         | mon_keyword_char NUMBER EOL
5642         | mon_keyword_char '-1' EOL
5643         | mon_keyword_grouping mon_group_list EOL
5644         ;
5645 mon_keyword_string : 'int_curr_symbol' | 'currency_symbol'
5646         | 'mon_decimal_point' | 'mon_thousands_sep'
5647         | 'positive_sign' | 'negative_sign'
5648         ;
5649 mon_string      : ''' char_list '''
5650         | ''''
5651         ;

```

```

5652     mon_keyword_char      : 'int_frac_digits' | 'frac_digits'
5653                           | 'p_cs_precedes' | 'p_sep_by_space'
5654                           | 'n_cs_precedes' | 'n_sep_by_space'
5655                           | 'p_sign_posn' | 'n_sign_posn'
5656                           ;

5657     mon_keyword_grouping  : 'mon_grouping'
5658                           ;

5659     mon_group_list        : NUMBER
5660                           | mon_group_list ';' NUMBER
5661                           ;

5662     monetary_tlr         : 'END' 'LC_MONETARY' EOL
5663                           ;

5664     /* The following is the LC_NUMERIC category grammar */
5665     lc_numeric            : numeric_hdr numeric_keywords      numeric_tlr
5666                           | numeric_hdr 'copy' locale_name EOL numeric_tlr
5667                           ;

5668     numeric_hdr           : 'LC_NUMERIC' EOL
5669                           ;

5670     numeric_keywords      : numeric_keywords numeric_keyword
5671                           | numeric_keyword
5672                           ;

5673     numeric_keyword       : num_keyword_string num_string EOL
5674                           | num_keyword_grouping num_group_list EOL
5675                           ;

5676     num_keyword_string    : 'decimal_point'
5677                           | 'thousands_sep'
5678                           ;

5679     num_string            : '"' char_list '"'
5680                           | '""'
5681                           ;

5682     num_keyword_grouping  : 'grouping'
5683                           ;

5684     num_group_list        : NUMBER
5685                           | num_group_list ';' NUMBER
5686                           ;

5687     numeric_tlr           : 'END' 'LC_NUMERIC' EOL
5688                           ;

5689     /* The following is the LC_TIME category grammar */
5690     lc_time               : time_hdr time_keywords          time_tlr
5691                           | time_hdr 'copy' locale_name EOL time_tlr
5692                           ;

5693     time_hdr              : 'LC_TIME' EOL
5694                           ;

```

```
5695     time_keywords      : time_keywords time_keyword
5696                          | time_keyword
5697                          ;
5698     time_keyword        : time_keyword_name time_list EOL
5699                          | time_keyword_fmt time_string EOL
5700                          | time_keyword_opt time_list EOL
5701                          ;
5702     time_keyword_name   : 'abday' | 'day' | 'abmon' | 'mon'
5703                          ;
5704     time_keyword_fmt    : 'd_t_fmt' | 'd_fmt' | 't_fmt'
5705                          | 'am_pm' | 't_fmt_ampm'
5706                          ;
5707     time_keyword_opt    : 'era' | 'era_d_fmt' | 'era_t_fmt'
5708                          | 'era_d_t_fmt' | 'alt_digits'
5709                          ;
5710     time_list           : time_list ';' time_string
5711                          | time_string
5712                          ;
5713     time_string         : '"' char_list '"'
5714                          ;
5715     time_tlr            : 'END' 'LC_TIME' EOL
5716                          ;
```

5717 **7.5 Locale Definition Example**

5718 The following is an example of a locale definition file that could be used as input to the *localedef*
 5719 utility. It assumes that the utility is executed with the *-f* option, naming a *charmap* file with (at
 5720 least) the following content:

```
5721     CHARMAP
5722     <space>      \x20
5723     <dollar>     \x24
5724     <A>         \101
5725     <a>         \141
5726     <A-acute>   \346
5727     <a-acute>   \365
5728     <A-grave>   \300
5729     <a-grave>   \366
5730     <b>         \142
5731     <C>         \103
5732     <c>         \143
5733     <c-cedilla> \347
5734     <d>         \x64
5735     <H>         \110
5736     <h>         \150
5737     <eszet>    \xb7
5738     <s>         \x73
5739     <z>         \x7a
5740     END CHARMAP
```

5741 It should not be taken as complete or to represent any actual locale, but only to illustrate the
 5742 syntax.

```
5743     #
5744     LC_CTYPE
5745     lower  <a>;<b>;<c>;<c-cedilla>;<d>;...;<z>
5746     upper  A;B;C;Ç;...;Z
5747     space  \x20;\x09;\x0a;\x0b;\x0c;\x0d
5748     blank  \040;\011
5749     toupper (<a>,<A>);(b,B);(c,C);(ç,Ç);(d,D);(z,Z)
5750     END LC_CTYPE
5751     #
5752     LC_COLLATE
5753     #
5754     # The following example of collation is based on
5755     # Canadian standard Z243.4.1-1998, "Canadian Alphanumeric
5756     # Ordering Standard For Character sets of CSA Z234.4 Standard".
5757     # (Other parts of this example locale definition file do not
5758     # purport to relate to Canada, or to any other real culture.)
5759     # The proposed standard defines a 4-weight collation, such that
5760     # in the first pass, characters are compared without regard to
5761     # case or accents; in second pass, backwards compare without
5762     # regard to case; in the third pass, forward compare without
5763     # regard to diacriticals. In the 3 first passes, non-alphabetic
5764     # characters are ignored; in the fourth pass, only special
5765     # characters are considered, such that "The string that has a
5766     # special character in the lowest position comes first. If two
```

```
5767      # strings have a special character in the same position, the
5768      # collation value of the special character determines ordering.
5769      #
5770      # Only a subset of the character set is used here; mostly to
5771      # illustrate the set-up.
5772      #
5773      collating-symbol <NULL>
5774      collating-symbol <LOW_VALUE>
5775      collating-symbol <LOWER-CASE>
5776      collating-symbol <SUBSCRIPT-LOWER>
5777      collating-symbol <SUPERSCRIPT-LOWER>
5778      collating-symbol <UPPER-CASE>
5779      collating-symbol <NO-ACCENT>
5780      collating-symbol <PECULIAR>
5781      collating-symbol <LIGATURE>
5782      collating-symbol <ACUTE>
5783      collating-symbol <GRAVE>
5784      # Further collating-symbols follow.
5785      #
5786      # Properly, the standard does not include any multi-character
5787      # collating elements; the one below is added for completeness.
5788      #
5789      collating_element <ch> from "<c><h>"
5790      collating_element <CH> from "<C><H>"
5791      collating_element <Ch> from "<C><h>"
5792      #
5793      order_start forward;backward;forward;forward,position
5794      #
5795      # Collating symbols are specified first in the sequence to allocate
5796      # basic collation values to them, lower than that of any character.
5797      <NULL>
5798      <LOW_VALUE>
5799      <LOWER-CASE>
5800      <SUBSCRIPT-LOWER>
5801      <SUPERSCRIPT-LOWER>
5802      <UPPER-CASE>
5803      <NO-ACCENT>
5804      <PECULIAR>
5805      <LIGATURE>
5806      <ACUTE>
5807      <GRAVE>
5808      <RING-ABOVE>
5809      <DIAERESIS>
5810      <TILDE>
5811      # Further collating symbols are given a basic collating value here.
5812      #
5813      # Here follow special characters.
5814      <space>          IGNORE;IGNORE;IGNORE;<space>
5815      # Other special characters follow here.
5816      #
5817      # Here follow the regular characters.
5818      <a>              <a>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
```



```

5819      <A>          <a>; <NO-ACCENT>; <UPPER-CASE>; IGNORE
5820      <a-acute>    <a>; <ACUTE>; <LOWER-CASE>; IGNORE
5821      <A-acute>    <a>; <ACUTE>; <UPPER-CASE>; IGNORE
5822      <a-grave>    <a>; <GRAVE>; <LOWER-CASE>; IGNORE
5823      <A-grave>    <a>; <GRAVE>; <UPPER-CASE>; IGNORE
5824      <ae>        "<a><e>"; "<LIGATURE><LIGATURE>"; \
5825                "<LOWER-CASE><LOWER-CASE>"; IGNORE
5826      <AE>        "<a><e>"; "<LIGATURE><LIGATURE>"; \
5827                "<UPPER-CASE><UPPER-CASE>"; IGNORE
5828      <b>          <b>; <NO-ACCENT>; <LOWER-CASE>; IGNORE
5829      <B>          <b>; <NO-ACCENT>; <UPPER-CASE>; IGNORE
5830      <c>          <c>; <NO-ACCENT>; <LOWER-CASE>; IGNORE
5831      <C>          <c>; <NO-ACCENT>; <UPPER-CASE>; IGNORE
5832      <ch>        <ch>; <NO-ACCENT>; <LOWER-CASE>; IGNORE
5833      <Ch>        <ch>; <NO-ACCENT>; <PECULIAR>; IGNORE
5834      <CH>        <ch>; <NO-ACCENT>; <UPPER-CASE>; IGNORE
5835      #
5836      # As an example, the strings "Bach" and "bach" could be encoded (for
5837      # compare purposes) as:
5838      # "Bach"    <b>; <a>; <ch>; <LOW_VALUE>; <NO_ACCENT>; <NO_ACCENT>; \
5839                <NO_ACCENT>; <LOW_VALUE>; <UPPER-CASE>; <LOWER-CASE>; \
5840                <LOWER-CASE>; <NULL>
5841      # "bach"   <b>; <a>; <ch>; <LOW_VALUE>; <NO_ACCENT>; <NO_ACCENT>; \
5842                <NO_ACCENT>; <LOW_VALUE>; <LOWER-CASE>; <LOWER-CASE>; \
5843                <LOWER-CASE>; <NULL>
5844      #
5845      # The two strings are equal in pass 1 and 2, but differ in pass 3.
5846      #
5847      # Further characters follow.
5848      #
5849      UNDEFINED    IGNORE; IGNORE; IGNORE; IGNORE
5850      #
5851      order_end
5852      #
5853      END LC_COLLATE
5854      #
5855      LC_MONETARY
5856      int_curr_symbol    "USD "
5857      currency_symbol    "$"
5858      mon_decimal_point  "."
5859      mon_grouping       3;0
5860      positive_sign      ""
5861      negative_sign      "- "
5862      p_cs_precedes      1
5863      n_sign_posn        0
5864      END LC_MONETARY
5865      #
5866      LC_NUMERIC
5867      copy "US_en.ASCII"
5868      END LC_NUMERIC
5869      #
5870      LC_TIME

```

```
5871      abday  "Sun";"Mon";"Tue";"Wed";"Thu";"Fri";"Sat"
5872      #
5873      day    "Sunday";"Monday";"Tuesday";"Wednesday";\
5874          "Thursday";"Friday";"Saturday"
5875      #
5876      abmon  "Jan";"Feb";"Mar";"Apr";"May";"Jun";\
5877          "Jul";"Aug";"Sep";"Oct";"Nov";"Dec"
5878      #
5879      mon    "January";"February";"March";"April";\
5880          "May";"June";"July";"August";"September";\
5881          "October";"November";"December"
5882      #
5883      d_t_fmt "%a %b %d %T %Z %Y\n"
5884      END LC_TIME
5885      #
5886      LC_MESSAGES
5887      yesexpr "^[yY][[:alpha:]]*"|(OK)"
5888      #
5889      noexpr  "^[nN][[:alpha:]]*"
5890      END LC_MESSAGES
```

Environment Variables

5891

5892 8.1 Environment Variable Definition

5893 Environment variables defined in this chapter affect the operation of multiple utilities, functions,
 5894 and applications. There are other environment variables that are of interest only to specific
 5895 utilities. Environment variables that apply to a single utility only are defined as part of the
 5896 utility description. See the ENVIRONMENT VARIABLES section of the utility descriptions in
 5897 the Shell and Utilities volume of IEEE Std. 1003.1-200x for information on environment variable
 5898 usage.

5899 The value of an environment variable is a string of characters. For a C-language program, an
 5900 array of strings called the environment is made available when a process begins. The array is
 5901 pointed to by the external variable *environ*, which is defined as:

```
5902     extern char **environ;
```

5903 These strings have the form *name=value*; *names* do not contain the character '='. For values to
 5904 be portable across systems conforming to IEEE Std. 1003.1-200x, the value shall be composed of
 5905 characters from the portable character set (except NUL and as indicated below). There is no
 5906 meaning associated with the order of strings in the environment. If more than one string in a
 5907 process' environment has the same *name*, the consequences are undefined.

5908 Environment variable names used by the utilities in the Shell and Utilities volume of
 5909 IEEE Std. 1003.1-200x shall consist solely of uppercase letters, digits, and the '_' (underscore)
 5910 from the characters defined in Table 6-1 (on page 133). Other characters may be permitted by an
 5911 implementation; applications shall tolerate the presence of such names. Uppercase and
 5912 lowercase letters retain their unique identities and are not folded together. The name space of
 5913 environment variable names containing lowercase letters is reserved for applications.
 5914 Applications can define any environment variables with names from this name space without
 5915 modifying the behavior of the standard utilities.

5916 The *values* that the environment variables may be assigned are not restricted except that they are
 5917 considered to end with a null byte and the total space used to store the environment and the
 5918 arguments to the process is limited to {ARG_MAX} bytes.

5919 Other *name=value* pairs may be placed in the environment by, for example, calling any of the
 5920 XSI *setenv()*, *unsetenv()*, or *putenv()* functions, manipulating the *environ* variable, or by using *envp*
 5921 arguments when creating a process; see *exec* in the System Interfaces volume of
 5922 IEEE Std. 1003.1-200x.

5923 It is unwise to conflict with certain variables that are frequently exported by widely used
 5924 command interpreters and applications:

5925
5926
5927
5928
5929
5930
5931
5932
5933
5934
5935
5936
5937
5938
5939
5940
5941
5942
5943
5944
5945

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

5946
5947
5948
5949
5950
5951
5952
5953
5954
5955
5956

If the variables in the following two sections are present in the environment during the execution of an application or utility, they are given the meaning described below. Some are placed into the environment by the implementation at the time the user logs in; all can be added or changed by the user or any ancestor of the current process. The implementation adds or changes environment variables named in IEEE Std. 1003.1-200x only as specified in IEEE Std. 1003.1-200x. If they are defined in the application's environment, the utilities in the Shell and Utilities volume of IEEE Std. 1003.1-200x and the functions in the System Interfaces volume of IEEE Std. 1003.1-200x assume they have the specified meaning. Conforming applications shall not set these environment variables to have meanings other than as described. See *getenv()* and the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.13, Shell Execution Environment for methods of accessing these variables.

5957 **8.2 Internationalization Variables**

5958 This section describes environment variables that are relevant to the operation of
 5959 internationalized interfaces described in the System Interfaces volume of IEEE Std. 1003.1-200x
 5960 and the Shell and Utilities volume of IEEE Std. 1003.1-200x.

5961 Users may use the following environment variables to announce specific localization
 5962 requirements to applications. Applications shall retrieve this information using the *setlocale()*
 5963 function to initialize the correct behavior of the internationalized interfaces. The descriptions of
 5964 the internationalization environment variables describe the resulting behavior only when the
 5965 application locale is initialized in this way.

5966 **LANG** This variable shall determine the locale category for native language, local
 5967 customs, and coded character set in the absence of the *LC_ALL* and other *LC_**
 5968 (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*,
 5969 *LC_TIME*) environment variables. This can be used by applications to
 5970 determine the language to use for error messages and instructions, collating
 5971 sequences, date formats, and so on.

5972 **LC_ALL** This variable shall determine the values for all locale categories. The value of
 5973 the *LC_ALL* environment variable has precedence over any of the other
 5974 environment variables starting with *LC_*(*LC_COLLATE*, *LC_CTYPE*,
 5975 *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*) and the *LANG*
 5976 environment variable.

5977 **LC_COLLATE** This variable shall determine the locale category for character collation. It
 5978 determines collation information for regular expressions and sorting,
 5979 including equivalence classes and multi-character collating elements, in
 5980 various utilities and the *strcoll()* and *strxfrm()* functions. Additional semantics
 5981 of this variable, if any, are implementation-defined.

5982 **LC_CTYPE** This variable shall determine the locale category for character handling
 5983 functions, such as *tolower()*, *toupper()*, and *isalpha()*. This environment
 5984 variable determines the interpretation of sequences of bytes of text data as
 5985 characters (for example, single as opposed to multi-byte characters), the
 5986 classification of characters (for example, alpha, digit, graph), and the behavior
 5987 of character classes. Additional semantics of this variable, if any, are
 5988 implementation-defined.

5989 **LC_MESSAGES** This variable shall determine the locale category for processing affirmative
 5990 and negative responses and the language and cultural conventions in which
 5991 XSI messages should be written. It also affects the behavior of the *catopen()*
 5992 function in determining the message catalog. Additional semantics of this
 5993 variable, if any, are implementation-defined. The language and cultural
 5994 conventions of diagnostic and informative messages whose format is
 5995 unspecified by IEEE Std. 1003.1-200x should be affected by the setting of
 5996 *LC_MESSAGES*.

5997 **LC_MONETARY** This variable shall determine the locale category for monetary-related numeric
 5998 formatting information. Additional semantics of this variable, if any, are
 5999 implementation-defined.

6000 **LC_NUMERIC** This variable shall determine the locale category for numeric formatting (for
 6001 example, thousands separator and radix character) information in various
 6002 utilities as well as the formatted I/O operations in *printf()* and *scanf()* and the
 6003 string conversion functions in *strtod()*. Additional semantics of this variable,
 6004 if any, are implementation-defined.

6005		<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.
6006			
6007			
6008	XSI	<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 substitution fields, a file name, and an optional suffix.
6009			
6010			
6011			
6012			For example:
6013			<code>NLSPATH="/system/nlslib/%N.cat"</code>
6014			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> (<i>%N</i>), with the suffix <code>.cat</code> .
6015			
6016			
6017			Substitution fields consist of a <code>'%'</code> symbol, followed by a single-letter keyword. The following keywords are currently defined:
6018			
6019			<code>%N</code> The value of the <i>name</i> parameter passed to <i>catopen()</i> .
6020			<code>%L</code> The value of the <i>LC_MESSAGES</i> category.
6021			<code>%l</code> The <i>language</i> element from the <i>LC_MESSAGES</i> category.
6022			<code>%t</code> The <i>territory</i> element from the <i>LC_MESSAGES</i> category.
6023			<code>%c</code> The <i>codeset</i> element from the <i>LC_MESSAGES</i> category.
6024			<code>%%</code> A single <code>'%'</code> character.
6025			An empty string is substituted if the specified value is not currently defined. The separators underscore (<code>'_'</code>) and period (<code>'.'</code>) are not included in <code>%t</code> and <code>%c</code> substitutions.
6026			
6027			
6028			Templates defined in <i>NLSPATH</i> are separated by colons (<code>':'</code>). A leading or two adjacent colons <code>": : "</code> is equivalent to specifying <code>%N</code> . For example:
6029			
6030			<code>NLSPATH=" : %N.cat : /nlslib/%L/%N.cat "</code>
6031			indicates to <i>catopen()</i> that it should look for the requested message catalog in <i>name</i> , <i>name.cat</i> , and <code>/nlslib/category/name.cat</code> , where <i>category</i> is the value of the <i>LC_MESSAGES</i> category of the current locale.
6032			
6033			
6034			Users should not set the <i>NLSPATH</i> variable unless they have a specific reason to override the default system path. Doing so causes undefined behavior in the standard utilities.
6035			
6036			
6037			The environment variables <i>LANG</i> , <i>LC_ALL</i> , <i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> , <i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i> , and <i>NLSPATH</i> provide for the support of internationalized applications. The standard utilities shall make use of these environment variables as described in this section and the individual ENVIRONMENT VARIABLES sections for the utilities. If these variables specify locale categories that are not based upon the same underlying codeset, the results are unspecified.
6038	XSI		
6039			
6040			
6041			
6042			
6043			The values of locale categories shall be determined by a precedence order; the first condition met below determines the value:
6044			
6045			1. If the <i>LC_ALL</i> environment variable is defined and is not null, the value of <i>LC_ALL</i> shall be used.
6046			

- 6047 2. If the *LC_** environment variable (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*,
6048 *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*) is defined and is not null, the value of the
6049 environment variable shall be used to initialize the category that corresponds to the
6050 environment variable.
- 6051 3. If the *LANG* environment variable is defined and is not null, the value of the *LANG*
6052 environment variable shall be used.
- 6053 4. If the *LANG* environment variable is not set or is set to the empty string, the
6054 implementation-defined default locale shall be used.

6055 If the locale value is "C" or "POSIX", the POSIX locale shall be used and the standard utilities
6056 behave in accordance with the rules in Section 7.2 (on page 144) for the associated category.

6057 If the locale value begins with a slash, it shall be interpreted as the path name of a file that was
6058 created in the output format used by the *localedef* utility; see OUTPUT FILES under *localedef*.
6059 Referencing such a path name results in that locale being used for the indicated category.

6060 XSI If the locale value has the form:

```
6061 language[_territory][.codeset]
```

6062 it refers to an implementation-provided locale, where settings of language, territory, and codeset
6063 are implementation-defined.

6064 *LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, and *LC_TIME* are
6065 defined to accept an additional field *@modifier*, which allows the user to select a specific instance
6066 of localization data within a single category (for example, for selecting the dictionary as opposed
6067 to the character ordering of data). The syntax for these environment variables is thus defined as:

```
6068 [language[_territory][.codeset][@modifier]]
```

6069 For example, if a user wanted to interact with the system in French, but required to sort German
6070 text files, *LANG* and *LC_COLLATE* could be defined as:

```
6071 LANG=Fr_FR  
6072 LC_COLLATE=De_DE
```

6073 This could be extended to select dictionary collation (say) by use of the *@modifier* field; for
6074 example:

```
6075 LC_COLLATE=De_DE@dict
```

6076

6077 An implementation may support other formats.

6078 If the locale value is not recognized by the implementation, the behavior is unspecified.

6079 At runtime, these values are bound to a program's locale by calling the *setlocale()* function.

6080 Additional criteria for determining a valid locale name are implementation-defined.

6081 **8.3 Other Environment Variables**

6082	<i>COLUMNS</i>	This variable shall represent a decimal integer >0 used to indicate the user's preferred width in column positions for the terminal screen or window; see Section 3.106 (on page 59). If this variable is unset or null, the implementation determines the number of columns, appropriate for the terminal or window, in an unspecified manner. When <i>COLUMNS</i> is set, any terminal-width information implied by <i>TERM</i> is overridden. Users and portable applications should not set <i>COLUMNS</i> unless they wish to override the system selection and produce output unrelated to the terminal characteristics.
6083		
6084		
6085		
6086		
6087		
6088		
6089		
6090		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.
6091		
6092		
6093	XSI <i>DATEMSK</i>	Indicates the path name of the template file used by <i>getdate()</i> .
6094	<i>HOME</i>	The system initializes this variable at the time of login to be a path name of the user's home directory. See < <i>pwd.h</i> >.
6095		
6096	<i>LINES</i>	This variable shall represent a decimal integer >0 used to indicate the user's preferred number of lines on a page or the vertical screen or window size in lines. A line in this case is a vertical measure large enough to hold the tallest character in the character set being displayed. If this variable is unset or null, the implementation determines the number of lines, appropriate for the 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 portable applications should not set <i>LINES</i> unless they wish to override the system selection and produce output unrelated to the terminal characteristics.
6097		
6098		
6099		
6100		
6101		
6102		
6103		
6104		
6105		
6106		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.
6107		
6108		
6109	<i>LOGNAME</i>	The system initializes 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 file name character set.
6110		
6111		
6112		
6113	XSI <i>MSGVERB</i>	Describes which message components shall be used in writing messages by <i>fmtmsg()</i> .
6114		
6115	<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 file name. The prefixes are separated by a colon (':'). When a non-zero-length prefix is applied to this file name, a slash is inserted between the prefix and the file name. 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 portable application shall use an actual path name (such as <i>.</i>) to represent the current working directory in <i>PATH</i> . The list is searched from beginning to end, applying the file name to each prefix, until an executable file with the specified name and appropriate execution permissions is found. If the path name being sought contains a slash, the search through the path prefixes is not performed. If the path name begins with a slash, the specified path is resolved (see Section 4.9 (on page 123)). If <i>PATH</i> is unset or is set to
6116		
6117		
6118		
6119		
6120		
6121		
6122		
6123		
6124		
6125		
6126		
6127		
6128		

6129		null, the path search is implementation-defined.
6130	<i>PWD</i>	This variable shall represent an absolute path name of the current working directory. It shall not contain any file name components of dot or dot-dot. The value is set by the <i>cd</i> utility.
6131		
6132		
6133	<i>SHELL</i>	This variable shall represent a path name 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.
6134		
6135		
6136		
6137		
6138	<i>TMPDIR</i>	This variable shall represent a path name of a directory made available for programs that need a place to create temporary files.
6139		
6140	<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.
6141		
6142		
6143		
6144	<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 for clarity):
6145		
6146		
6147		
6148		
6149		<i>:characters</i>
6150		or:
6151		<i>std offset dst offset, rule</i>
6152		If <i>TZ</i> is of the first format (that is, if the first character is a colon), the characters following the colon are handled in an implementation-defined manner.
6153		
6154		
6155		The expanded format (for all <i>TZs</i> whose value does not have a colon as the first character) is as follows:
6156		
6157		<i>stdoffset[dst[offset][,start[/time],end[/time]]]</i>
6158		Where:
6159	<i>std</i> and <i>dst</i>	Indicate no less than three, nor more than {TZNAME_MAX}, bytes that are the designation for the standard (<i>std</i>) or the alternative (<i>dst</i> —such as Daylight Savings Time) timezone. Only <i>std</i> is required; if <i>dst</i> is missing, then the alternative time does not apply in this locale.
6160		
6161		
6162		
6163		
6164		Each of these fields may occur in either of two formats quoted or unquoted:
6165		
6166		— In the quoted form, the first character shall be the less-than ('<') character and the last character shall be the greater-than ('>') character. All characters between these quoting characters shall be alphanumeric characters in the current locale, the plus-sign ('+') character, or the minus-sign ('-') character. The <i>std</i> and <i>dst</i> fields in this case do not include the quoting characters.
6167		
6168		
6169		
6170		
6171		
6172		

6173		— In the unquoted form, all characters in these fields shall be
6174		alphabetic characters in the current locale.
6175		The interpretation of these fields is unspecified if either field is
6176		less than three bytes (except for the case when <i>dst</i> is missing),
6177		more than {TZNAME_MAX} bytes, or if they contain characters
6178		other than those specified.
6179	<i>offset</i>	Indicates the value added to the local time to arrive at
6180		Coordinated Universal Time. The <i>offset</i> has the form:
6181		<i>hh</i> [: <i>mm</i> [: <i>ss</i>]]
6182		The minutes (<i>mm</i>) and seconds (<i>ss</i>) are optional. The hour (<i>hh</i>)
6183		shall be required and may be a single digit. The <i>offset</i> following
6184		<i>std</i> shall be required. If no <i>offset</i> follows <i>dst</i> , the alternative time
6185		is assumed to be one hour ahead of standard time. One or more
6186		digits may be used; the value is always interpreted as a decimal
6187		number. The hour shall be between zero and 24, and the minutes
6188		(and seconds)—if present—between zero and 59. The result of
6189		using values outside of this range is unspecified. If preceded by
6190		a '-', the timezone shall be east of the Prime Meridian;
6191		otherwise, it shall be west (which may be indicated by an
6192		optional preceding '+').
6193	<i>rule</i>	Indicates when to change to and back from the alternative time.
6194		The <i>rule</i> has the form:
6195		<i>date</i> [/ <i>time</i>] , <i>date</i> [/ <i>time</i>]
6196		where the first <i>date</i> describes when the change from standard to
6197		alternative time occurs and the second <i>date</i> describes when the
6198		change back happens. Each <i>time</i> field describes when, in current
6199		local time, the change to the other time is made.
6200		The format of <i>date</i> is one of the following:
6201	<i>Jn</i>	The Julian day <i>n</i> ($1 \leq n \leq 365$). Leap days shall not be
6202		counted. That is, in all years—including leap years—
6203		February 28 is day 59 and March 1 is day 60. It is
6204		impossible to refer explicitly to the occasional February
6205		29.
6206	<i>n</i>	The zero-based Julian day ($0 \leq n \leq 365$). Leap days shall
6207		be counted, and it is possible to refer to February 29.
6208	<i>Mm.n.d</i>	The <i>d</i> 'th day ($0 \leq d \leq 6$) of week <i>n</i> of month <i>m</i> of the
6209		year ($1 \leq n \leq 5$, $1 \leq m \leq 12$, where week 5 means "the
6210		last <i>d</i> day in month <i>m</i> " which may occur in either the
6211		fourth or the fifth week). Week 1 is the first week in
6212		which the <i>d</i> 'th day occurs. Day zero is Sunday.
6213		The <i>time</i> has the same format as <i>offset</i> except that no leading sign
6214		('-' or '+') is allowed. The default, if <i>time</i> is not given, shall be
6215		02:00:00.
6216		

Regular Expressions

6217

6218 *Regular Expressions* (REs) provide a mechanism to select specific strings from a set of character
6219 strings.

6220 Regular expressions are a context-independent syntax that can represent a wide variety of
6221 character sets and character set orderings, where these character sets are interpreted according
6222 to the current locale. While many regular expressions can be interpreted differently depending
6223 on the current locale, many features, such as character class expressions, provide for contextual
6224 invariance across locales.

6225 The Basic Regular Expression (BRE) notation and construction rules in Section 9.3 (on page 198)
6226 shall apply to most utilities supporting regular expressions. Some utilities, instead, support the
6227 Extended Regular Expressions (ERE) described in Section 9.4 (on page 203); any exceptions for
6228 both cases are noted in the descriptions of the specific utilities using regular expressions. Both
6229 BREs and EREs are supported by the Regular Expression Matching interface in the System
6230 Interfaces volume of IEEE Std. 1003.1-200x under *regcomp()*, *regexexec()*, and related functions. |

6231 **9.1 Regular Expression Definitions**

6232 For the purposes of this section, the following definitions shall apply:

6233 **entire regular expression**6234 The concatenated set of one or more BREs or EREs that make up the pattern specified for
6235 string selection.6236 **matched**6237 A sequence of zero or more characters shall be said to be matched by a BRE or ERE when
6238 the characters in the sequence correspond to a sequence of characters defined by the
6239 pattern.6240 Matching shall be based on the bit pattern used for encoding the character, not on the
6241 graphic representation of the character. This means that if a character set contains two or
6242 more encodings for a graphic symbol, or if the strings searched contain text encoded in
6243 more than one codeset, no attempt is made to search for any other representation of the
6244 encoded symbol. If that is required, the user can specify equivalence classes containing all
6245 variations of the desired graphic symbol.6246 The search for a matching sequence starts at the beginning of a string and stops when the
6247 first sequence matching the expression is found, where *first* is defined to mean “begins
6248 earliest in the string”. If the pattern permits a variable number of matching characters and
6249 thus there is more than one such sequence starting at that point, the longest such sequence
6250 is matched. For example: the BRE "bb*" matches the second to fourth characters of *abbbc*,
6251 and the ERE *(wee | week)(knights | night)* matches all ten characters of *weeknights*.6252 Consistent with the whole match being the longest of the leftmost matches, each subpattern,
6253 from left to right, shall match the longest possible string. For this purpose, a null string shall
6254 be considered to be longer than no match at all. For example, matching the BRE
6255 "\(.*\).*" against "abcdef", the subexpression "(\1)" is "abcdef", and matching
6256 the BRE "\(a*\).*" against "bc", the subexpression "(\1)" is the null string.6257 When a multi-character collating element in a bracket expression (see Section 9.3.5 (on page
6258 199)) is involved, the longest sequence shall be measured in characters consumed from the
6259 string to be matched; that is, the collating element counts not as one element, but as the
6260 number of characters it matches.6261 **BRE (ERE) matching a single character**

6262 A BRE or ERE that shall match either a single character or a single collating element.

6263 Only a BRE or ERE of this type that includes a bracket expression (see Section 9.3.5 (on page
6264 199)) can match a collating element.6265 **BRE (ERE) matching multiple characters**

6266 A BRE or ERE that shall match a concatenation of single characters or collating elements.

6267 Such a BRE or ERE is made up from a BRE (ERE) matching a single character and BRE (ERE)
6268 special characters.6269 **invalid**6270 This section uses the term *invalid* for certain constructs or conditions. Invalid REs shall
6271 cause the utility or function using the RE to generate an error condition. When *invalid* is not
6272 used, violations of the specified syntax or semantics for REs produce undefined results: this
6273 may entail an error, enabling an extended syntax for that RE, or using the construct in error
6274 as literal characters to be matched. For example, the BRE construct "\{1,2,3\}" does not
6275 comply with the grammar. A portable application cannot rely on it producing an error nor
6276 matching the literal characters "\{1,2,3\}".

6277 9.2 Regular Expression General Requirements

6278 The requirements in this section shall apply to both basic and extended regular expressions.

6279 The use of regular expressions is generally associated with text processing. REs (BREs and EREs)
6280 operate on text strings; that is, zero or more characters followed by an end-of-string delimiter
6281 (typically NUL). Some utilities employing regular expressions limit the processing to lines; that
6282 is, zero or more characters followed by a <newline> character. In the regular expression
6283 processing described in IEEE Std. 1003.1-200x, the <newline> character is regarded as an
6284 ordinary character and both a period and a non-matching list can match one. The Shell and
6285 Utilities volume of IEEE Std. 1003.1-200x specifies within the individual descriptions of those
6286 standard utilities employing regular expressions whether they permit matching of <newline>
6287 characters; if not stated otherwise, the use of literal <newline> characters or any escape sequence
6288 equivalent produces undefined results. Those utilities (like *grep*) that do not allow <newline>
6289 characters to match are responsible for eliminating any <newline> character from strings before
6290 matching against the RE. The *regcomp()* function in the System Interfaces volume of
6291 IEEE Std. 1003.1-200x, however, can provide support for such processing without violating the
6292 rules of this section.

6293 The interfaces specified in IEEE Std. 1003.1-200x do not permit the inclusion of a NUL character
6294 in an RE or in the string to be matched. If during the operation of a standard utility a NUL is
6295 included in the text designated to be matched, that NUL may designate the end of the text string
6296 for the purposes of matching.

6297 When a standard utility or function that uses regular expressions specifies that pattern matching
6298 shall be performed without regard to the case (uppercase or lowercase) of either data or
6299 patterns, then when each character in the string is matched against the pattern, not only the
6300 character, but also its case counterpart (if any), shall be matched. This definition of case-
6301 insensitive processing is intended to allow matching of multi-character collating elements as
6302 well as characters. For example, as each character in the string is matched using both its cases,
6303 the RE "[[.Ch.]]" when matched against the string "char", is in reality matched against
6304 "ch", "Ch", "cH", and "CH".

6305 The implementation shall support any regular expression that does not exceed 256 bytes in
6306 length.

6307 **9.3 Basic Regular Expressions**6308 **9.3.1 BREs Matching a Single Character or Collating Element**

6309 A BRE ordinary character, a special character preceded by a backslash or a period, shall match a
 6310 single character. A bracket expression shall match a single character or a single collating
 6311 element.

6312 **9.3.2 BRE Ordinary Characters**

6313 An ordinary character is a BRE that matches itself: any character in the supported character set,
 6314 except for the BRE special characters listed in Section 9.3.3.

6315 The interpretation of an ordinary character preceded by a backslash ('**') is undefined, except
 6316 for:

- 6317 • The characters '**', '*(*', '*{*', and '*}*'
- 6318 • The digits 1 to 9 inclusive (see Section 9.3.6 (on page 201))
- 6319 • A character inside a bracket expression

6320 **9.3.3 BRE Special Characters**

6321 A *BRE special character* has special properties in certain contexts. Outside those contexts, or when
 6322 preceded by a backslash, such a character is a BRE that matches the special character itself. The
 6323 BRE special characters and the contexts in which they have their special meaning are as follows:

6324 . [** The period, left-bracket, and backslash shall be special except when used in a bracket
 6325 expression (see Section 9.3.5 (on page 199)). An expression containing a '*[*' that is not
 6326 preceded by a backslash and is not part of a bracket expression produces undefined
 6327 results.

6328 * The asterisk shall be special except when used:

- 6329 • In a bracket expression
- 6330 • As the first character of an entire BRE (after an initial '*^*', if any)
- 6331 • As the first character of a subexpression (after an initial '*^*', if any); see Section
 6332 9.3.6 (on page 201)

6333 ^ The circumflex shall be special when used as:

- 6334 • An anchor (see Section 9.3.8 (on page 202))
- 6335 • The first character of a bracket expression (see Section 9.3.5 (on page 199))

6336 \$ The dollar sign shall be special when used as an anchor.

6337 **9.3.4 Periods in BREs**

6338 A period ('*.*'), when used outside a bracket expression, is a BRE that shall match any character
 6339 in the supported character set except NUL.

6340 **9.3.5 RE Bracket Expression**

6341 A bracket expression (an expression enclosed in square brackets, "[]") is an RE that matches a
 6342 single collating element contained in the non-empty set of collating elements represented by the
 6343 bracket expression.

6344 The following rules and definitions apply to bracket expressions:

6345 1. A *bracket expression* is either a matching list expression or a non-matching list expression. It
 6346 consists of one or more expressions: collating elements, collating symbols, equivalence
 6347 classes, character classes, or range expressions. Portable applications shall not use range
 6348 expressions, even though all implementations shall support them. The right-bracket (']')
 6349 shall lose its special meaning and represents itself in a bracket expression if it occurs first in
 6350 the list (after an initial circumflex ('^'), if any). Otherwise, it shall terminate the bracket
 6351 expression, unless it appears in a collating symbol (such as "[.].]") or is the ending
 6352 right-bracket for a collating symbol, equivalence class, or character class. The special
 6353 characters '.', '*', '[', and '\' (period, asterisk, left-bracket, and backslash,
 6354 respectively) shall lose their special meaning within a bracket expression.

6355 The character sequences "[.", "[=", and "[:" (left-bracket followed by a period, equals-
 6356 sign, or colon) shall be special inside a bracket expression and are used to delimit collating
 6357 symbols, equivalence class expressions, and character class expressions. These symbols
 6358 shall be followed by a valid expression and the matching terminating sequence ".]",
 6359 " =]", or " :]", as described in the following items.

6360 2. A *matching list* expression specifies a list that shall match any one of the expressions
 6361 represented in the list. The first character in the list shall not be the circumflex; for
 6362 example, "[abc]" is an RE that matches any of the characters 'a', 'b', or 'c'.

6363 3. A *non-matching list* expression begins with a circumflex ('^'), and specifies a list that shall
 6364 match any character or collating element except for the expressions represented in the list
 6365 after the leading circumflex. For example, "[^abc]" is an RE that matches any character
 6366 or collating element except the characters 'a', 'b', or 'c'. The circumflex shall have this
 6367 special meaning only when it occurs first in the list, immediately following the left-bracket.

6368 4. A *collating symbol* is a collating element enclosed within bracket-period ("[.]" and ".]")
 6369 delimiters. Collating elements are defined as described in Section 7.3.2.4 (on page 158).
 6370 Portable applications shall represent multi-character collating elements as collating
 6371 symbols when it is necessary to distinguish them from a list of the individual characters
 6372 that make up the multi-character collating element. For example, if the string "ch" is a
 6373 collating element in the current collation sequence with the associated collating symbol
 6374 <ch>, the expression "[[.ch.]]" shall be treated as an RE matching the character
 6375 sequence 'ch', while "[ch]" shall be treated as an RE matching 'c' or 'h'. Collating
 6376 symbols are recognized only inside bracket expressions. This implies that the RE
 6377 "[[.ch.]]*c" shall match the first to fifth character in the string "chchch". If the string
 6378 is not a collating element in the current collating sequence definition, or if the collating
 6379 element has no characters associated with it (for example, see the symbol <HIGH> in the
 6380 example collation definition shown in Section 7.3.2.2 (on page 157)), the symbol shall be
 6381 treated as an invalid expression.

6382 5. An *equivalence class expression* shall represent the set of collating elements belonging to an
 6383 equivalence class, as described in Section 7.3.2.4 (on page 158). Only primary equivalence
 6384 classes shall be recognized. The class shall be expressed by enclosing any one of the
 6385 collating elements in the equivalence class within bracket-equal ("[=" and "=]")
 6386 delimiters. For example, if 'a', 'â', and 'ä' belong to the same equivalence class, then
 6387 "[[=a=]b]", "[[=â=]b]", and "[[=â=]b]" are each equivalent to "[aââb]". If the

6388 collating element does not belong to an equivalence class, the equivalence class expression
6389 shall be treated as a *collating symbol*.

6390 6. A *character class expression* shall represent the set of characters belonging to a character
6391 class, as defined in the *LC_CTYPE* category in the current locale. All character classes
6392 specified in the current locale shall be recognized. A character class expression is expressed
6393 as a character class name enclosed within bracket-colon ("[:]" and "[:]") delimiters.

6394 The following character class expressions shall be supported in all locales:

```
6395      [:alnum:]      [:cntrl:]      [:lower:]      [:space:]
6396      [:alpha:]     [:digit:]     [:print:]     [:upper:]
6397      [:blank:]     [:graph:]     [:punct:]     [:xdigit:]
```

6398 XSI In addition, character class expressions of the form:

```
6399      [:name:]
```

6400 are recognized in those locales where the *name* keyword has been given a **charclass**
6401 definition in the *LC_CTYPE* category.

6402 7. A range expression represents the set of collating elements that fall between two elements
6403 in the collating element order of the current locale, inclusive. A range expression shall be
6404 expressed as the starting point and the ending point separated by a hyphen ('-').

6405 Range expressions shall not be used in portable applications because their behavior is
6406 dependent on the collating sequence.

6407 In the following, all examples assume the collation sequence specified for the POSIX locale,
6408 unless another collation sequence is specifically defined.

6409 The starting range point and the ending range point shall be a collating element or
6410 collating symbol. An equivalence class expression used as a starting or ending point of a
6411 range expression produces unspecified results. An equivalence class can be used portably
6412 within a bracket expression, but only outside the range. For example, the unspecified
6413 expression "[[=e=]-f]" should be given as "[[=e=]e-f]". The ending range point
6414 shall collate equal to or higher than the starting range point; otherwise, the expression is
6415 treated as invalid. The order used is the order in which the collating elements are specified
6416 in the current collation definition. One-to-many mappings (see the description of
6417 *LC_COLLATE* in Section 7.3.2 (on page 155)) are not performed. For example, assuming
6418 that the character eszet ('ß') is placed in the collation sequence after 'r' and 's', but
6419 before 't' and that it maps to the sequence "ss" for collation purposes, then the
6420 expression "[r-s]" matches only 'r' and 's', but the expression "[s-t]" matches
6421 's', 'ß', or 't'.

6422 The interpretation of range expressions where the ending range point is also the starting
6423 range point of a subsequent range expression (for example, "[a-m-o]") is undefined.

6424 The hyphen character shall be treated as itself if it occurs first (after an initial '^', if any)
6425 or last in the list, or as an ending range point in a range expression. As examples, the
6426 expressions "[-ac]" and "[ac-]" are equivalent and match any of the characters 'a',
6427 'c', or '-'; "[^ac-]" and "[^ac-]" are equivalent and match any characters except
6428 'a', 'c', or '-'; the expression "[%-]" matches any of the characters between '%' and
6429 '-' inclusive; the expression "[--@]" matches any of the characters between '-' and
6430 '@' inclusive; and the expression "[a--@]" is invalid, because the letter 'a' follows the
6431 symbol '-' in the POSIX locale. To use a hyphen as the starting range point, it shall either
6432 come first in the bracket expression or be specified as a collating symbol; for example,
6433 "[[][-.-]-0]", which matches either a right bracket or any character or collating element

6434 that collates between hyphen and 0, inclusive.

6435 If a bracket expression specifies both '-' and ']', the ']' shall be placed first (after the
6436 '^', if any) and the '-' last within the bracket expression.

6437 9.3.6 BREs Matching Multiple Characters

6438 The following rules can be used to construct BREs matching multiple characters from BREs
6439 matching a single character:

6440 1. The concatenation of BREs shall match the concatenation of the strings matched by each
6441 component of the BRE.

6442 2. A *subexpression* can be defined within a BRE by enclosing it between the character pairs
6443 "\" and "\". Such a subexpression shall match whatever it would have matched
6444 without the "\" and "\" , except that anchoring within subexpressions is optional
6445 behavior; see Section 9.3.8 (on page 202). Subexpressions can be arbitrarily nested.

6446 3. The *back-reference* expression '\n' shall match the same (possibly empty) string of
6447 characters as was matched by a subexpression enclosed between "\" and "\"
6448 preceding the '\n'. The character 'n' shall be a digit from 1 through 9, specifying the
6449 *n*th subexpression (the one that begins with the *n*th "\" from the beginning of the
6450 pattern and ends with the corresponding paired "\"). The expression is invalid if less
6451 than *n* subexpressions precede the '\n'. For example, the expression "\(.*)\1\$" |
6452 matches a line consisting of two adjacent appearances of the same string, and the
6453 expression "\(a\)*\1" fails to match 'a'. When the referenced subexpression matched
6454 more than one string, the back-referenced expression shall refer to the last matched string.
6455 If the subexpression referenced by the back-reference matches more than one string
6456 because of an asterisk ('*') or an interval expression (see item (5)), the back-reference
6457 shall match the last (rightmost) of these strings.

6458 4. When a BRE matching a single character, a subexpression, or a back-reference is followed
6459 by the special character asterisk ('*'), together with that asterisk it shall match what zero
6460 or more consecutive occurrences of the BRE would match. For example, "[ab]*" and
6461 "[ab][ab]" are equivalent when matching the string "ab".

6462 5. When a BRE matching a single character, a subexpression, or a back-reference is followed
6463 by an *interval expression* of the format "{m}", "{m,}", or "{m,n}", together with
6464 that interval expression it shall match what repeated consecutive occurrences of the BRE
6465 would match. The values of *m* and *n* are decimal integers in the range 0
6466 $\leq m \leq n \leq \{RE_DUP_MAX\}$, where *m* specifies the exact or minimum number of occurrences
6467 and *n* specifies the maximum number of occurrences. The expression "{m}" shall match
6468 exactly *m* occurrences of the preceding BRE, "{m,}" shall match at least *m* occurrences,
6469 and "{m,n}" shall match any number of occurrences between *m* and *n*, inclusive.

6470 For example, in the string "abababcccccd" the BRE "c{3}" is matched by
6471 characters '7' to '9', the BRE "\(ab)\{4,}" is not matched at all, and the BRE
6472 "c{1,3}d" is matched by characters ten to thirteen.

6473 The behavior of multiple adjacent duplication symbols ('*' and intervals) produces undefined
6474 results.

6475 A subexpression repeated by an asterisk ('*') or an interval expression shall not match a null
6476 expression unless this is the only match for the repetition or it is necessary to satisfy the exact or
6477 minimum number of occurrences for the interval expression.

6478 **9.3.7 BRE Precedence**

6479 The order of precedence shall be as shown in the following table:

BRE Precedence (from high to low)	
6480	
6481	Collation-related bracket symbols [=] [: :] [. .]
6482	Escaped characters \<special character>
6483	Bracket expression []
6484	Subexpressions/back-references \ (\) \ n
6485	Single-character-BRE duplication * \ { m , n \ }
6486	Concatenation
6487	Anchoring ^ \$

6488 **9.3.8 BRE Expression Anchoring**

6489 A BRE can be limited to matching strings that begin or end a line; this is called *anchoring*. The
 6490 circumflex and dollar sign special characters shall be considered BRE anchors in the following
 6491 contexts:

- 6492 1. A circumflex (' ^ ') shall be an anchor when used as the first character of an entire BRE.
 6493 The implementation may treat the circumflex as an anchor when used as the first character
 6494 of a subexpression. The circumflex shall anchor the expression (or optionally
 6495 subexpression) to the beginning of a string; only sequences starting at the first character of
 6496 a string shall be matched by the BRE. For example, the BRE " ^ ab " matches " ab " in the
 6497 string " abcdef ", but fails to match in the string " cdefab ". The BRE " \ (^ ab \) " may
 6498 match the former string. A portable BRE shall escape a leading circumflex in a
 6499 subexpression to match a literal circumflex.
- 6500 2. A dollar sign (' \$ ') shall be an anchor when used as the last character of an entire BRE.
 6501 The implementation may treat a dollar sign as an anchor when used as the last character of
 6502 a subexpression. The dollar sign shall anchor the expression (or optionally subexpression)
 6503 to the end of the string being matched; the dollar sign can be said to match the end-of-
 6504 string following the last character.
- 6505 3. A BRE anchored by both ' ^ ' and ' \$ ' shall match only an entire string. For example, the
 6506 BRE " ^ abcdef \$ " matches strings consisting only of " abcdef ".

6507 9.4 Extended Regular Expressions

6508 The *extended regular expression* (ERE) notation and construction rules shall apply to utilities
6509 defined as using extended regular expressions; any exceptions to the following rules are noted in
6510 the descriptions of the specific utilities using EREs.

6511 9.4.1 EREs Matching a Single Character or Collating Element

6512 An ERE ordinary character, a special character preceded by a backslash, or a period shall match
6513 a single character. A bracket expression shall match a single character or a single collating
6514 element. An *ERE matching a single character* enclosed in parentheses shall match the same as the
6515 ERE without parentheses would have matched.

6516 9.4.2 ERE Ordinary Characters

6517 An *ordinary character* is an ERE that matches itself. An ordinary character is any character in the
6518 supported character set, except for the ERE special characters listed in Section 9.4.3. The
6519 interpretation of an ordinary character preceded by a backslash ('\') is undefined.

6520 9.4.3 ERE Special Characters

6521 An *ERE special character* has special properties in certain contexts. Outside those contexts, or
6522 when preceded by a backslash, such a character shall be an ERE that matches the special
6523 character itself. The extended regular expression special characters and the contexts in which
6524 they shall have their special meaning are as follows:

6525 . [\ (The period, left-bracket, backslash, and left-parenthesis shall be special except when
6526 used in a bracket expression (see Section 9.3.5 (on page 199)). Outside a bracket
6527 expression, a left-parenthesis immediately followed by a right-parenthesis produces
6528 undefined results.

6529) The right-parenthesis shall be special when matched with a preceding left-parenthesis,
6530 both outside a bracket expression.

6531 * + ? { The asterisk, plus-sign, question-mark, and left-brace shall be special except when used
6532 in a bracket expression (see Section 9.3.5 (on page 199)). Any of the following uses
6533 produce undefined results:

- 6534 • If these characters appear first in an ERE, or immediately following a vertical-line,
6535 circumflex, or left-parenthesis

- 6536 • If a left-brace is not part of a valid interval expression (see Section 9.4.6 (on page
6537 204))

6538 | The vertical-line is special except when used in a bracket expression (see Section 9.3.5
6539 (on page 199)). A vertical-line appearing first or last in an ERE, or immediately
6540 following a vertical-line or a left-parenthesis, or immediately preceding a right-
6541 parenthesis, produces undefined results.

6542 ^ The circumflex shall be special when used as:

- 6543 • An anchor (see Section 9.4.9 (on page 205))

- 6544 • The first character of a bracket expression (see Section 9.3.5 (on page 199))

6545 \$ The dollar sign shall be special when used as an anchor.

6546 **9.4.4 Periods in EREs**

6547 A period ('.'), when used outside a bracket expression, is an ERE that shall match any
6548 character in the supported character set except NUL.

6549 **9.4.5 ERE Bracket Expression**

6550 The rules for ERE Bracket Expressions are the same as for Basic Regular Expressions; see Section
6551 9.3.5 (on page 199).

6552 **9.4.6 EREs Matching Multiple Characters**

6553 The following rules shall be used to construct EREs matching multiple characters from EREs
6554 matching a single character:

- 6555 1. A *concatenation of EREs* shall match the concatenation of the character sequences matched
6556 by each component of the ERE. A concatenation of EREs enclosed in parentheses shall
6557 match whatever the concatenation without the parentheses matches. For example, both the
6558 ERE "cd" and the ERE "(cd)" are matched by the third and fourth character of the string
6559 "abcdefabcdef".
- 6560 2. When an ERE matching a single character or an ERE enclosed in parentheses is followed by
6561 the special character plus-sign ('+'), together with that plus-sign it shall match what one
6562 or more consecutive occurrences of the ERE would match. For example, the ERE
6563 "b+(bc)" matches the fourth to seventh characters in the string "acabbbbcde". And,
6564 "[ab]+" and "[ab][ab]*" are equivalent.
- 6565 3. When an ERE matching a single character or an ERE enclosed in parentheses is followed by
6566 the special character asterisk ('*'), together with that asterisk it shall match what zero or
6567 more consecutive occurrences of the ERE would match. For example, the ERE "b*c"
6568 matches the first character in the string "cabbbbcde", and the ERE "b*cd" matches the
6569 third to seventh characters in the string "cabbbbcdebbbbbbbcdbbc". And, "[ab]*" and
6570 "[ab][ab]" are equivalent when matching the string "ab".
- 6571 4. When an ERE matching a single character or an ERE enclosed in parentheses is followed by
6572 the special character question-mark ('?'), together with that question-mark it shall match
6573 what zero or one consecutive occurrences of the ERE would match. For example, the ERE
6574 "b?c" matches the second character in the string "acabbbbcde".
- 6575 5. When an ERE matching a single character or an ERE enclosed in parentheses is followed by
6576 an *interval expression* of the format "{m}", "{m,}", or "{m,n}", together with that
6577 interval expression it shall match what repeated consecutive occurrences of the ERE would
6578 match. The values of *m* and *n* are decimal integers in the range $0 \leq m \leq n \leq \text{RE_DUP_MAX}$,
6579 where *m* specifies the exact or minimum number of occurrences and *n* specifies the
6580 maximum number of occurrences. The expression "{m}" matches exactly *m* occurrences
6581 of the preceding ERE, "{m,}" matches at least *m* occurrences, and "{m,n}" matches any
6582 number of occurrences between *m* and *n*, inclusive.

6583 For example, in the string "abababcccccd" the ERE "c{3}" is matched by characters
6584 '7' to '9' and the ERE "(ab){2,}" is matched by characters one to six.

6585 The behavior of multiple adjacent duplication symbols ('+', '*', '?', and intervals) produces
6586 undefined results.

6587 An ERE matching a single character repeated by an '*', '?', or an interval expression shall not
6588 match a null expression unless this is the only match for the repetition or it is necessary to satisfy
6589 the exact or minimum number of occurrences for the interval expression.

6590 **9.4.7 ERE Alternation**

6591 Two EREs separated by the special character vertical-line ('|') shall match a string that is
 6592 matched by either. For example, the ERE "a((bc)|d)" matches the string "abc" and the string
 6593 "ad". Single characters, or expressions matching single characters, separated by the vertical bar
 6594 and enclosed in parentheses, shall be treated as an ERE matching a single character.

6595 **9.4.8 ERE Precedence**

6596 The order of precedence shall be as shown in the following table:

ERE Precedence (from high to low)	
6598 Collation-related bracket symbols	[==] [::] [..]
6599 Escaped characters	\<special character>
6600 Bracket expression	[]
6601 Grouping	()
6602 Single-character-ERE duplication	* + ? {m,n}
6603 Concatenation	
6604 Anchoring	^ \$
6605 Alternation	

6606 For example, the ERE "abba|cde" matches either the string "abba" or the string "cde"
 6607 (rather than the string "abbade" or "abbcde", because concatenation has a higher order of
 6608 precedence than alternation).

6609 **9.4.9 ERE Expression Anchoring**

6610 An ERE can be limited to matching strings that begin or end a line; this is called *anchoring*. The
 6611 circumflex and dollar sign special characters shall be considered ERE anchors when used
 6612 anywhere outside a bracket expression. This shall have the following effects:

- 6613 1. A circumflex ('^') outside a bracket expression shall anchor the expression or
 6614 subexpression it begins to the beginning of a string; such an expression or subexpression
 6615 can match only a sequence starting at the first character of a string. For example, the EREs
 6616 "^ab" and "(^ab)" match "ab" in the string "abcdef", but fail to match in the string
 6617 "cdefab", and the ERE "a^b" is valid, but can never match because the 'a' prevents the
 6618 expression "^b" from matching starting at the first character.
- 6619 2. A dollar sign ('\$') outside a bracket expression shall anchor the expression or
 6620 subexpression it ends to the end of a string; such an expression or subexpression can
 6621 match only a sequence ending at the last character of a string. For example, the EREs
 6622 "ef\$" and "(ef\$)" match "ef" in the string "abcdef", but fail to match in the string
 6623 "cdefab", and the ERE "e\$f" is valid, but can never match because the 'f' prevents the
 6624 expression "e\$" from matching ending at the last character.

6625 **9.5 Regular Expression Grammar**

6626 Grammars describing the syntax of both basic and extended regular expressions are presented in
 6627 this section. The grammar takes precedence over the text. See the Shell and Utilities volume of
 6628 IEEE Std. 1003.1-200x, Section 1.10, Grammar Conventions.

6629 **9.5.1 BRE/ERE Grammar Lexical Conventions**

6630 The lexical conventions for regular expressions are as described in this section.

6631 Except as noted, the longest possible token or delimiter beginning at a given point is recognized.

6632 The following tokens are processed (in addition to those string constants shown in the
 6633 grammar):

6634 **COLL_ELEM** Any single-character collating element, unless it is a META_CHAR.

6635 **BACKREF** Applicable only to basic regular expressions. The character string
 6636 consisting of '\ ' followed by a single-digit numeral, '1' to '9'.

6637 **DUP_COUNT** Represents a numeric constant. It shall be an integer in the range 0
 6638 ≤DUP_COUNT ≤{RE_DUP_MAX}. This token is only recognized when
 6639 the context of the grammar requires it. At all other times, digits not
 6640 preceded by '\ ' are treated as ORD_CHAR.

6641 **META_CHAR** One of the characters:

6642 ^ When found first in a bracket expression

6643 - When found anywhere but first (after an initial '^', if any) or
 6644 last in a bracket expression, or as the ending range point in a
 6645 range expression

6646] When found anywhere but first (after an initial '^', if any) in a
 6647 bracket expression

6648 **L_ANCHOR** Applicable only to basic regular expressions. The character '^' when it
 6649 appears as the first character of a basic regular expression and when not
 6650 QUOTED_CHAR. The '^' may be recognized as an anchor elsewhere;
 6651 see Section 9.3.8 (on page 202).

6652 **ORD_CHAR** A character, other than one of the special characters in SPEC_CHAR.

6653 **QUOTED_CHAR** In a BRE, one of the character sequences:

6654 \^ \. * \[\\$ \\

6655 In an ERE, one of the character sequences:

6656 \^ \. \[\\$ \(\) \|
 6657 * \+ \? \{ \\

6658 **R_ANCHOR** (Applicable only to basic regular expressions.) The character '\$' when it
 6659 appears as the last character of a basic regular expression and when not
 6660 QUOTED_CHAR. The '\$' may be recognized as an anchor elsewhere;
 6661 see Section 9.3.8 (on page 202).

6662 **SPEC_CHAR** For basic regular expressions, one of the following special characters:

6663 . Anywhere outside bracket expressions

6664 \ Anywhere outside bracket expressions

```

6665      [      Anywhere outside bracket expressions
6666      ^      When used as an anchor (see Section 9.3.8 (on page 202)) or
6667           when first in a bracket expression
6668      $      When used as an anchor
6669      *      Anywhere except first in an entire RE, anywhere in a bracket
6670           expression, directly following "\(", directly following an
6671           anchoring '^'
6672
6673      For extended regular expressions, shall be one of the following special
6674           characters found anywhere outside bracket expressions:
6675
6676           ^      .      [      $      (      )      |
6677           *      +      ?      {      \

```

(The close-parenthesis shall be considered special in this context only if matched with a preceding open-parenthesis.)

6678 9.5.2 RE and Bracket Expression Grammar

6679 This section presents the grammar for basic regular expressions, including the bracket
6680 expression grammar that is common to both BREs and EREs.

```

6681 %token   ORD_CHAR QUOTED_CHAR DUP_COUNT
6682 %token   BACKREF L_ANCHOR R_ANCHOR
6683 %token   Back_open_paren  Back_close_paren
6684 /*      '\('      '\)'      */
6685 %token   Back_open_brace  Back_close_brace
6686 /*      '\{'      '\}'      */
6687 /* The following tokens are for the Bracket Expression
6688    grammar common to both REs and EREs. */
6689 %token   COLL_ELEM META_CHAR
6690 %token   Open_equal Equal_close Open_dot Dot_close Open_colon Colon_close
6691 /*      '['      '='      '['      '.'      '['      ':'      ':'      */
6692 %token   class_name
6693 /* class_name is a keyword to the LC_CTYPE locale category */
6694 /* (representing a character class) in the current locale */
6695 /* and is only recognized between [: and :] */
6696 %start   basic_reg_exp
6697 %%
6698 /* -----
6699    Basic Regular Expression
6700    -----
6701 */
6702 basic_reg_exp :      RE_expression
6703              | L_ANCHOR
6704              |      R_ANCHOR
6705              | L_ANCHOR      R_ANCHOR
6706              | L_ANCHOR RE_expression
6707              |      RE_expression R_ANCHOR

```

```

6708         | L_ANCHOR RE_expression R_ANCHOR
6709         ;
6710 RE_expression :          simple_RE
6711         | RE_expression simple_RE
6712         ;
6713 simple_RE      : nondupl_RE
6714         | nondupl_RE RE_dupl_symbol
6715         ;
6716 nondupl_RE     : one_character_RE
6717         | Back_open_paren RE_expression Back_close_paren
6718         | BACKREF
6719         ;
6720 one_character_RE : ORD_CHAR
6721         | QUOTED_CHAR
6722         | '.'
6723         | bracket_expression
6724         ;
6725 RE_dupl_symbol : '*'
6726         | Back_open_brace DUP_COUNT          Back_close_brace
6727         | Back_open_brace DUP_COUNT ','      Back_close_brace
6728         | Back_open_brace DUP_COUNT ',' DUP_COUNT Back_close_brace
6729         ;

6730 /* -----
6731    Bracket Expression
6732    -----
6733 */
6734 bracket_expression : '[' matching_list ']'
6735         | '[' nonmatching_list ']'
6736         ;
6737 matching_list      : bracket_list
6738         ;
6739 nonmatching_list   : '^' bracket_list
6740         ;
6741 bracket_list       : follow_list
6742         | follow_list '-'
6743         ;
6744 follow_list        :          expression_term
6745         | follow_list expression_term
6746         ;
6747 expression_term    : single_expression
6748         | range_expression
6749         ;
6750 single_expression  : end_range
6751         | character_class
6752         | equivalence_class
6753         ;
6754 range_expression   : start_range end_range
6755         | start_range '-'
6756         ;
6757 start_range        : end_range '-'
6758         ;
6759 end_range          : COLL_ELEM

```



```

6760         | collating_symbol
6761         ;
6762 collating_symbol : Open_dot COLL_ELEM Dot_close
6763         | Open_dot META_CHAR Dot_close
6764         ;
6765 equivalence_class : Open_equal COLL_ELEM Equal_close
6766         ;
6767 character_class : Open_colon class_name Colon_close
6768         ;

```

6769 The BRE grammar does not permit L_ANCHOR or R_ANCHOR inside "\(" and "\)" (which
6770 implies that '^' and '\$' are ordinary characters). This reflects the semantic limits on the
6771 application, as noted in Section 9.3.8 (on page 202). Implementations are permitted to extend the
6772 language to interpret '^' and '\$' as anchors in these locations, and as such, portable
6773 applications cannot use unescaped '^' and '\$' in positions inside "\(" and "\)" that might
6774 be interpreted as anchors.

6775 9.5.3 ERE Grammar

6776 This section presents the grammar for extended regular expressions, excluding the bracket
6777 expression grammar.

6778 **Note:** The bracket expression grammar and the associated %token lines are identical
6779 between BREs and EREs. It has been omitted from the ERE section to avoid
6780 unnecessary editorial duplication.

```

6781 %token ORD_CHAR QUOTED_CHAR DUP_COUNT
6782 %start extended_reg_exp
6783 %%
6784 /* -----
6785    Extended Regular Expression
6786    -----
6787 */
6788 extended_reg_exp : ERE_branch
6789         | extended_reg_exp '|' ERE_branch
6790         ;
6791 ERE_branch : ERE_expression
6792         | ERE_branch ERE_expression
6793         ;
6794 ERE_expression : one_character_ERE
6795         | '^'
6796         | '$'
6797         | '(' extended_reg_exp ')'
6798         | ERE_expression ERE_dupl_symbol
6799         ;
6800 one_character_ERE : ORD_CHAR
6801         | QUOTED_CHAR
6802         | '.'
6803         | bracket_expression
6804         ;
6805 ERE_dupl_symbol : '*'
6806         | '+'
6807         | '?'
6808         | '{' DUP_COUNT '}'

```

```

6809         | '{' DUP_COUNT ',' '}'
6810         | '{' DUP_COUNT ',' DUP_COUNT '}'
6811         ;

```

6812 The ERE grammar does not permit several constructs that previous sections specify as having
 6813 undefined results:

- 6814 • ORD_CHAR preceded by '\'
- 6815 • One or more *ERE_dupl_symbols* appearing first in an ERE, or immediately following '|',
 6816 '^', or '('
- 6817 • '{' not part of a valid *ERE_dupl_symbol*
- 6818 • '|' appearing first or last in an ERE, or immediately following '|' or '(', or immediately
 6819 preceding ')'

6820 Implementations are permitted to extend the language to allow these. Portable applications |
 6821 cannot use such constructs.

Directory Structure and Devices

6822

6823 10.1 Directory Structure and Files

6824 The following directories shall exist on conforming systems and portable applications shall
 6825 make use of them only as described. Portable applications shall not assume the ability to create
 6826 files in any of these directories, unless specified below.

6827 / The root directory.

6828 /dev Contains /dev/console, /dev/null, and /dev/tty, described below.

6829 The following directory shall exist on conforming systems and shall be used as described.

6830 /tmp A directory made available for programs that need a place to create temporary
 6831 files. Applications are allowed to create files in this directory, but cannot assume
 6832 that such files are preserved between invocations of the application.

6833 The following files shall exist on conforming systems and shall be both readable and writable.

6834 /dev/null An infinite data source and data sink. Data written to /dev/null shall be discarded.
 6835 Reads from /dev/null shall always return end-of-file (EOF).

6836 /dev/tty In each process, a synonym for the controlling terminal associated with the process
 6837 group of that process, if any. It is useful for programs or shell procedures that wish
 6838 to be sure of writing messages to or reading data from the terminal no matter how
 6839 output has been redirected. It can also be used for programs that demand the name
 6840 of a file for output, when typed output is desired and it is tiresome to find out
 6841 what terminal is currently in use.

6842 The following file shall exist on conforming systems and need not be readable or writable:

6843 /dev/console The /dev/console file is a generic name given to the system console. It is usually
 6844 linked to a particular machine-dependent special file. It shall provide a basic I/O
 6845 interface to the system console.

6846 10.2 Output Devices and Terminal Types

6847 The utilities in the Shell and Utilities volume of IEEE Std. 1003.1-200x historically have been
 6848 implemented on a wide range of terminal types, but a conforming implementation need not
 6849 support all features of all utilities on every conceivable terminal. IEEE Std. 1003.1-200x states
 6850 which features are optional for certain classes of terminals in the individual utility description
 6851 sections. The implementation shall document which terminal types it supports and which of
 6852 these features and utilities are not supported by each terminal.

6853 When a feature or utility is not supported on a specific terminal type, as allowed by
 6854 IEEE Std. 1003.1-200x, and the implementation considers such a condition to be an error
 6855 preventing use of the feature or utility, the implementation shall indicate such conditions
 6856 through diagnostic messages or exit status values or both (as appropriate to the specific utility
 6857 description) that inform the user that the terminal type lacks the appropriate capability.

6858 IEEE Std. 1003.1-200x uses a notational convention based on historical practice that identifies
 6859 some of the control characters defined in Section 7.3.1 (on page 147) in a manner easily

6860 remembered by users on many terminals. The correspondence between this “<control>-char”
 6861 notation and the actual control characters is shown in the following table. When
 6862 IEEE Std. 1003.1-200x refers to a character by its <control>- name, it is referring to the actual
 6863 control character shown in the Value column of the table, which is not necessarily the exact
 6864 control key sequence on all terminals. Some terminals have keyboards that do not allow the
 6865 direct transmission of all the non-alphanumeric characters shown. In such cases, the system
 6866 documentation shall describe which data sequences transmitted by the terminal are interpreted
 6867 by the system as representing the special characters.

6868 **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>-\ <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>-?		

6886 **Note:** The notation uses uppercase letters for arbitrary editorial reasons. There is no
 6887 implication that the keystrokes represent control-shift-letter sequences.

General Terminal Interface

6888

6889 This chapter describes a general terminal interface that shall be provided. It shall be supported
6890 on any asynchronous communications ports if the implementation provides them. It is
6891 implementation-defined whether it supports network connections or synchronous ports, or
6892 both.

6893 11.1 Interface Characteristics

6894 11.1.1 Opening a Terminal Device File

6895 When a terminal device file is opened, it normally causes the thread to wait until a connection is
6896 established. In practice, application programs seldom open these files; they are opened by
6897 special programs and become an application's standard input, output, and error files.

6898 As described in *open()*, opening a terminal device file with the `O_NONBLOCK` flag clear shall
6899 cause the thread to block until the terminal device is ready and available. If `CLOCAL` mode is
6900 not set, this means blocking until a connection is established. If `CLOCAL` mode is set in the
6901 terminal, or the `O_NONBLOCK` flag is specified in the *open()*, the *open()* function shall return a
6902 file descriptor without waiting for a connection to be established.

6903 11.1.2 Process Groups

6904 A terminal may have a foreground process group associated with it. This foreground process
6905 group plays a special role in handling signal-generating input characters, as discussed in Section
6906 11.1.9 (on page 217).

6907 A command interpreter process supporting job control can allocate the terminal to different jobs,
6908 or process groups, by placing related processes in a single process group and associating this
6909 process group with the terminal. A terminal's foreground process group may be set or examined
6910 by a process, assuming the permission requirements are met; see *tcgetpgrp()* and *tcsetpgrp()*. The
6911 terminal interface aids in this allocation by restricting access to the terminal by processes that are
6912 not in the current process group; see Section 11.1.4 (on page 214).

6913 When there is no longer any process whose process ID or process group ID matches the process
6914 group ID of the foreground process group, the terminal shall have no foreground process group.
6915 It is unspecified whether the terminal has a foreground process group when there is a process
6916 whose process ID matches the foreground process ID, but whose process group ID does not. No
6917 actions defined in IEEE Std. 1003.1-200x, other than allocation of a controlling terminal or a
6918 successful call to *tcsetpgrp()*, cause a process group to become the foreground process group of
6919 the terminal.

6920 11.1.3 The Controlling Terminal

6921 A terminal may belong to a process as its controlling terminal. Each process of a session that has
6922 a controlling terminal has the same controlling terminal. A terminal may be the controlling
6923 terminal for at most one session. The controlling terminal for a session is allocated by the session
6924 leader in an implementation-defined manner. If a session leader has no controlling terminal, and
6925 opens a terminal device file that is not already associated with a session without using the
6926 O_NOCTTY option (see *open()*), it is implementation-defined whether the terminal becomes the
6927 controlling terminal of the session leader. If a process which is not a session leader opens a
6928 terminal file, or the O_NOCTTY option is used on *open()*, then that terminal shall not become
6929 the controlling terminal of the calling process. When a controlling terminal becomes associated
6930 with a session, its foreground process group shall be set to the process group of the session
6931 leader.

6932 The controlling terminal is inherited by a child process during a *fork()* function call. A process
6933 relinquishes its controlling terminal when it creates a new session with the *setsid()* function;
6934 other processes remaining in the old session that had this terminal as their controlling terminal
6935 continue to have it. Upon the close of the last file descriptor in the system (whether or not it is in
6936 the current session) associated with the controlling terminal, it is unspecified whether all
6937 processes that had that terminal as their controlling terminal cease to have any controlling
6938 terminal. Whether and how a session leader can reacquire a controlling terminal after the
6939 controlling terminal has been relinquished in this fashion is unspecified. A process does not
6940 relinquish its controlling terminal simply by closing all of its file descriptors associated with the
6941 controlling terminal if other processes continue to have it open.

6942 When a controlling process terminates, the controlling terminal is dissociated from the current
6943 session, allowing it to be acquired by a new session leader. Subsequent access to the terminal by
6944 other processes in the earlier session may be denied, with attempts to access the terminal treated
6945 as if a modem disconnect had been sensed.

6946 11.1.4 Terminal Access Control

6947 If a process is in the foreground process group of its controlling terminal, read operations shall
6948 be allowed, as described in Section 11.1.5 (on page 215). Any attempts by a process in a
6949 background process group to read from its controlling terminal cause its process group to be
6950 sent a SIGTTIN signal unless one of the following special cases applies: if the reading process is
6951 ignoring or blocking the SIGTTIN signal, or if the process group of the reading process is
6952 orphaned, the *read()* returns -1 , with *errno* set to [EIO] and no signal is sent. The default action of
6953 the SIGTTIN signal is to stop the process to which it is sent. See <**signal.h**>.

6954 If a process is in the foreground process group of its controlling terminal, write operations shall
6955 be allowed as described in Section 11.1.8 (on page 217). Attempts by a process in a background
6956 process group to write to its controlling terminal shall cause the process group to be sent a
6957 SIGTTOU signal unless one of the following special cases applies: if TOSTOP is not set, or if
6958 TOSTOP is set and the process is ignoring or blocking the SIGTTOU signal, the process is
6959 allowed to write to the terminal and the SIGTTOU signal is not sent. If TOSTOP is set, and the
6960 process group of the writing process is orphaned, and the writing process is not ignoring or
6961 blocking the SIGTTOU signal, the *write()* returns -1 , with *errno* set to [EIO] and no signal is sent.

6962 Certain calls that set terminal parameters are treated in the same fashion as *write()*, except that
6963 TOSTOP is ignored; that is, the effect is identical to that of terminal writes when TOSTOP is set
6964 (see Section 11.2.5 (on page 223), *tcdrain()*, *tcfLOW()*, *tcfLush()*, *tcsendbreak()*, *tcsetattr()*, and
6965 *tcsetpgrp()*).

6966 11.1.5 Input Processing and Reading Data

6967 A terminal device associated with a terminal device file may operate in full-duplex mode, so that
 6968 data may arrive even while output is occurring. Each terminal device file has an *input queue*,
 6969 associated with it, into which incoming data is stored by the system before being read by a
 6970 process. The system may impose a limit, {MAX_INPUT}, on the number of bytes that may be
 6971 stored in the input queue. The behavior of the system when this limit is exceeded is
 6972 implementation-defined.

6973 Two general kinds of input processing are available, determined by whether the terminal device
 6974 file is in canonical mode or non-canonical mode. These modes are described in Section 11.1.6 and
 6975 Section 11.1.7 (on page 216). Additionally, input characters are processed according to the
 6976 *c_iflag* (see Section 11.2.2 (on page 219)) and *c_lflag* (see Section 11.2.5 (on page 223)) fields.
 6977 Such processing can include *echoing*, which in general means transmitting input characters
 6978 immediately back to the terminal when they are received from the terminal. This is useful for
 6979 terminals that can operate in full-duplex mode.

6980 The manner in which data is provided to a process reading from a terminal device file is
 6981 dependent on whether the terminal file is in canonical or non-canonical mode, and on whether
 6982 or not the O_NONBLOCK flag is set by *open()* or *fcntl()*.

6983 If the O_NONBLOCK flag is clear, then the read request shall be blocked until data is available
 6984 or a signal has been received. If the O_NONBLOCK flag is set, then the read request shall be
 6985 completed, without blocking, in one of three ways:

- 6986 1. If there is enough data available to satisfy the entire request, the *read()* shall complete
 6987 successfully and shall return the number of bytes read.
- 6988 2. If there is not enough data available to satisfy the entire request, the *read()* shall complete
 6989 successfully, having read as much data as possible, and shall return the number of bytes it
 6990 was able to read.
- 6991 3. If there is no data available, the *read()* shall return -1 , with *errno* set to [EAGAIN].

6992 When data is available depends on whether the input processing mode is canonical or non-
 6993 canonical. The following sections, Section 11.1.6 and Section 11.1.7 (on page 216), describe each
 6994 of these input processing modes.

6995 11.1.6 Canonical Mode Input Processing

6996 In canonical mode input processing, terminal input is processed in units of lines. A line is
 6997 delimited by a newline character (NL), an end-of-file character (EOF), or an end-of-line (EOL)
 6998 character. See Section 11.1.9 (on page 217) for more information on EOF and EOL. This means
 6999 that a read request shall not return until an entire line has been typed or a signal has been
 7000 received. Also, no matter how many bytes are requested in the *read()* call, at most one line shall
 7001 be returned. It is not, however, necessary to read a whole line at once; any number of bytes, even
 7002 one, may be requested in a *read()* without losing information.

7003 If {MAX_CANON} is defined for this terminal device, it is a limit on the number of bytes in a
 7004 line. The behavior of the system when this limit is exceeded is implementation-defined. If
 7005 {MAX_CANON} is not defined, there is no such limit; see *pathconf()*.

7006 Erase and kill processing occur when either of two special characters, the ERASE and KILL
 7007 characters (see Section 11.1.9 (on page 217)), is received. This processing affects data in the input
 7008 queue that has not yet been delimited by a newline (NL), EOF, or EOL character. This un-
 7009 delimited data makes up the current line. The ERASE character deletes the last character in the
 7010 current line, if there is one. The KILL character deletes all data in the current line, if there are any.
 7011 The ERASE and KILL characters have no effect if there is no data in the current line. The ERASE

7012 and KILL characters themselves are not placed in the input queue.

7013 11.1.7 Non-Canonical Mode Input Processing

7014 In non-canonical mode input processing, input bytes are not assembled into lines, and erase and
7015 kill processing do not occur. The values of the MIN and TIME members of the `c_cc` array are
7016 used to determine how to process the bytes received. The IEEE Std. 1003.1-200x does not specify
7017 whether the setting of `O_NONBLOCK` takes precedence over MIN or TIME settings. Therefore,
7018 if `O_NONBLOCK` is set, `read()` may return immediately, regardless of the setting of MIN or
7019 TIME. Also, if no data is available, `read()` may either return 0, or return `-1` with `errno` set to
7020 `[EAGAIN]`.

7021 MIN represents the minimum number of bytes that should be received when the `read()` function
7022 returns successfully. TIME is a timer of 0.1 second granularity that is used to time out bursty and
7023 short-term data transmissions. If MIN is greater than `{MAX_INPUT}`, the response to the request
7024 is undefined. The four possible values for MIN and TIME and their interactions are described
7025 below.

7026 Case A: MIN>0, TIME>0

7027 In case A, TIME serves as an inter-byte timer and is activated after the first byte is received. Since
7028 it is an inter-byte timer, it is reset after a byte is received. The interaction between MIN and
7029 TIME is as follows. As soon as one byte is received, the inter-byte timer is started. If MIN bytes
7030 are received before the inter-byte timer expires (remember that the timer is reset upon receipt of
7031 each byte), the read is satisfied. If the timer expires before MIN bytes are received, the characters
7032 received to that point are returned to the user. Note that if TIME expires at least one byte is
7033 returned because the timer would not have been enabled unless a byte was received. In this case
7034 (MIN>0, TIME>0) the read shall block until the MIN and TIME mechanisms are activated by the
7035 receipt of the first byte, or a signal is received. If the data is in the buffer at the time of the `read()`,
7036 the result shall be as if the data has been received immediately after the `read()`.

7037 Case B: MIN>0, TIME=0

7038 In case B, since the value of TIME is zero, the timer plays no role and only MIN is significant. A
7039 pending read is not satisfied until MIN bytes are received (that is, the pending read shall block
7040 until MIN bytes are received), or a signal is received. A program that uses case B to read record-
7041 based terminal I/O may block indefinitely in the read operation.

7042 Case C: MIN=0, TIME>0

7043 In case C, since MIN=0, TIME no longer represents an inter-byte timer. It now serves as a read
7044 timer that is activated as soon as the `read()` function is processed. A read is satisfied as soon as a
7045 single byte is received or the read timer expires. Note that in case C if the timer expires, no bytes
7046 are returned. If the timer does not expire, the only way the read can be satisfied is if a byte is
7047 received. If bytes are not received, the read shall not block indefinitely waiting for a byte; if no
7048 byte is received within $TIME * 0.1$ seconds after the read is initiated, the `read()` returns a value of
7049 zero, having read no data. If the data is in the buffer at the time of the `read()`, the timer shall be
7050 started as if the data has been received immediately after the `read()`.

7051 **Case D: MIN=0, TIME=0**

7052 The minimum of either the number of bytes requested or the number of bytes currently available
7053 shall be returned without waiting for more bytes to be input. If no characters are available, *read()*
7054 shall return a value of zero, having read no data.

7055 **11.1.8 Writing Data and Output Processing**

7056 When a process writes one or more bytes to a terminal device file, they are processed according
7057 to the *c_oflag* field (see Section 11.2.3 (on page 220)). The implementation may provide a
7058 buffering mechanism; as such, when a call to *write()* completes, all of the bytes written have
7059 been scheduled for transmission to the device, but the transmission has not necessarily
7060 completed. See *write()* for the effects of *O_NONBLOCK* on *write()*.

7061 **11.1.9 Special Characters**

7062 Certain characters have special functions on input or output or both. These functions are
7063 summarized as follows:

7064 **INTR** Special character on input, which is recognized if the *ISIG* flag is set. Generates a
7065 *SIGINT* signal which is sent to all processes in the foreground process group for which
7066 the terminal is the controlling terminal. If *ISIG* is set, the *INTR* character is discarded
7067 when processed.

7068 **QUIT** Special character on input, which is recognized if the *ISIG* flag is set. Generates a
7069 *SIGQUIT* signal which is sent to all processes in the foreground process group for
7070 which the terminal is the controlling terminal. If *ISIG* is set, the *QUIT* character is
7071 discarded when processed.

7072 **ERASE** Special character on input, which is recognized if the *ICANON* flag is set. Erases the
7073 last character in the current line; see Section 11.1.6 (on page 215). It shall not erase
7074 beyond the start of a line, as delimited by an *NL*, *EOF*, or *EOL* character. If *ICANON* is
7075 set, the *ERASE* character is discarded when processed.

7076 **KILL** Special character on input, which is recognized if the *ICANON* flag is set. Deletes the
7077 entire line, as delimited by an *NL*, *EOF*, or *EOL* character. If *ICANON* is set, the *KILL*
7078 character is discarded when processed.

7079 **EOF** Special character on input, which is recognized if the *ICANON* flag is set. When
7080 received, all the bytes waiting to be read are immediately passed to the process without
7081 waiting for a newline, and the *EOF* is discarded. Thus, if there are no bytes waiting
7082 (that is, the *EOF* occurred at the beginning of a line), a byte count of zero shall be
7083 returned from the *read()*, representing an end-of-file indication. If *ICANON* is set, the
7084 *EOF* character is discarded when processed.

7085 **NL** Special character on input, which is recognized if the *ICANON* flag is set. It is the line
7086 delimiter newline. It cannot be changed.

7087 **EOL** Special character on input, which is recognized if the *ICANON* flag is set. It is an
7088 additional line delimiter, like *NL*.

7089 **SUSP** If the *ISIG* flag is set, receipt of the *SUSP* character causes a *SIGTSTP* signal to be sent
7090 to all processes in the foreground process group for which the terminal is the
7091 controlling terminal, and the *SUSP* character is discarded when processed.

7092 **STOP** Special character on both input and output, which is recognized if the *IXON* (output
7093 control) or *IXOFF* (input control) flag is set. Can be used to suspend output
7094 temporarily. It is useful with CRT terminals to prevent output from disappearing

7095 before it can be read. If IXON is set, the STOP character is discarded when processed.

7096 START Special character on both input and output, which is recognized if the IXON (output
7097 control) or IXOFF (input control) flag is set. Can be used to resume output that has
7098 been suspended by a STOP character. If IXON is set, the START character is discarded
7099 when processed.

7100 CR Special character on input, which is recognized if the ICANON flag is set; it is the
7101 carriage-return character. When ICANON and ICRNL are set and IGNCR is not set,
7102 this character is translated into an NL, and has the same effect as an NL character.

7103 The NL and CR characters cannot be changed. It is implementation-defined whether the START
7104 and STOP characters can be changed. The values for INTR, QUIT, ERASE, KILL, EOF, EOL, and
7105 SUSP shall be changeable to suit individual tastes. Special character functions associated with
7106 changeable special control characters can be disabled individually.

7107 If two or more special characters have the same value, the function performed when that
7108 character is received is undefined.

7109 A special character is recognized not only by its value, but also by its context; for example, an
7110 implementation may support multi-byte sequences that have a meaning different from the
7111 meaning of the bytes when considered individually. Implementations may also support
7112 additional single-byte functions. These implementation-defined multi-byte or single-byte
7113 functions are recognized only if the IEXTEN flag is set; otherwise, data is received without
7114 interpretation, except as required to recognize the special characters defined in this section.

7115 XSI If IEXTEN is set, the ERASE, KILL, and EOF characters can be escaped by a preceding '\'
7116 character, in which case no special function occurs.

7117 11.1.10 Modem Disconnect

7118 If a modem disconnect is detected by the terminal interface for a controlling terminal, and if
7119 CLOCAL is not set in the **c_cflag** field for the terminal (see Section 11.2.4 (on page 222)), the
7120 SIGHUP signal is sent to the controlling process for which the terminal is the controlling
7121 terminal. Unless other arrangements have been made, this causes the controlling process to
7122 terminate (see *exit()*). Any subsequent read from the terminal device shall return the value of
7123 zero, indicating end-of-file; see *read()*. Thus, processes that read a terminal file and test for end-
7124 of-file can terminate appropriately after a disconnect. If the EIO condition as specified in *read()*
7125 also exists, it is unspecified whether on EOF condition or the [EIO] is returned. Any subsequent
7126 *write()* to the terminal device returns -1 , with *errno* set to [EIO], until the device is closed.

7127 11.1.11 Closing a Terminal Device File

7128 The last process to close a terminal device file shall cause any output to be sent to the device and
7129 any input to be discarded. If HUPCL is set in the control modes and the communications port
7130 supports a disconnect function, the terminal device shall perform a disconnect.

7131 11.2 Parameters that Can be Set

7132 11.2.1 The termios Structure

7133 Routines that need to control certain terminal I/O characteristics shall do so by using the
7134 **termios** structure as defined in the `<termios.h>` header. The members of this structure include
7135 (but are not limited to):

Member Type	Array Size	Member Name	Description
<code>tcflag_t</code>		<code>c_iflag</code>	Input modes.
<code>tcflag_t</code>		<code>c_oflag</code>	Output modes.
<code>tcflag_t</code>		<code>c_cflag</code>	Control modes.
<code>tcflag_t</code>		<code>c_lflag</code>	Local modes.
<code>cc_t</code>	NCCS	<code>c_cc[]</code>	Control characters.

7143 The types `tcflag_t` and `cc_t` are defined in the `<termios.h>` header. They shall be unsigned
7144 integer types.

7145 11.2.2 Input Modes

7146 Values of the `c_iflag` field describe the basic terminal input control, and are composed of the
7147 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name
7148 symbols in this table are defined in `<termios.h>`:

Mask Name	Description
BRKINT	Signal interrupt on break.
ICRNL	Map CR to NL on input.
IGNBRK	Ignore break condition.
IGNCR	Ignore CR.
IGNPAR	Ignore characters with parity errors.
INLCR	Map NL to CR on input.
INPCK	Enable input parity check.
ISTRIP	Strip character.
IXANY	Enable any character to restart output.
IXOFF	Enable start/stop input control.
IXON	Enable start/stop output control.
PARMRK	Mark parity errors.

7163 In the context of asynchronous serial data transmission, a break condition is defined as a
7164 sequence of zero-valued bits that continues for more than the time to send one byte. The entire
7165 sequence of zero-valued bits is interpreted as a single break condition, even if it continues for a
7166 time equivalent to more than one byte. In contexts other than asynchronous serial data
7167 transmission, the definition of a break condition is implementation-defined.

7168 If IGNBRK is set, a break condition detected on input is ignored; that is, not put on the input
7169 queue and therefore not read by any process. If IGNBRK is not set and BRKINT is set, the break
7170 condition shall flush the input and output queues, and if the terminal is the controlling terminal
7171 of a foreground process group, the break condition shall generate a single SIGINT signal to that
7172 foreground process group. If neither IGNBRK nor BRKINT is set, a break condition is read as a
7173 single 0x00, or if PARMRK is set, as 0xff 0x00 0x00.

7174 If IGNPAR is set, a byte with a framing or parity error (other than break) is ignored.

7175 If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than
7176 break) is given to the application as the three-byte sequence 0xff 0x00 X, where 0xff 0x00 is a
7177 two-byte flag preceding each sequence and X is the data of the byte received in error. To avoid
7178 ambiguity in this case, if ISTRIP is not set, a valid byte of 0xff is given to the application as 0xff
7179 0xff. If neither PARMRK nor IGNPAR is set, a framing or parity error (other than break) is given
7180 to the application as a single byte 0x00.

7181 If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is
7182 disabled, allowing output parity generation without input parity errors. Note that whether input
7183 parity checking is enabled or disabled is independent of whether parity detection is enabled or
7184 disabled (see Section 11.2.4 (on page 222)). If parity detection is enabled but input parity
7185 checking is disabled, the hardware to which the terminal is connected shall recognize the parity
7186 bit, but the terminal special file shall not check whether or not this bit is correctly set.

7187 If ISTRIP is set, valid input bytes are first stripped to seven bits; otherwise, all eight bits are
7188 processed.

7189 If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a
7190 received CR character is ignored (not read). If IGNCR is not set and ICRNL is set, a received CR
7191 character is translated into an NL character.

7192 XSI If IXANY is set, any input character shall restart output that has been suspended.

7193 If IXON is set, start/stop output control is enabled. A received STOP character shall suspend
7194 output and a received START character shall restart output. When IXON is set, START and
7195 STOP characters are not read, but merely perform flow control functions. When IXON is not set,
7196 the START and STOP characters are read.

7197 If IXOFF is set, start/stop input control is enabled. The system shall transmit STOP characters,
7198 which are intended to cause the terminal device to stop transmitting data, as needed to prevent
7199 the input queue from overflowing and causing implementation-defined behavior, and shall
7200 transmit START characters, which are intended to cause the terminal device to resume
7201 transmitting data, as soon as the device can continue transmitting data without risk of
7202 overflowing the input queue. The precise conditions under which STOP and START characters
7203 are transmitted are implementation-defined.

7204 The initial input control value after *open()* is implementation-defined.

7205 11.2.3 Output Modes

7206 The **c_oflag** field specifies the terminal interface's treatment of output, and is composed of the
7207 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name
7208 symbols in this table are defined in `<termios.h>`:

7209
7210
7211
7212
7213
7214
7215
7216
7217
7218
7219
7220
7221
7222
7223
7224
7225
7226
7227
7228
7229
7230
7231
7232
7233
7234
7235
7236
7237
7238
7239

XSI

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.

7240
7241
7242

7243
7244
7245
7246
7247
7248
7249

7250
7251
7252
7253
7254

7255

7256
7257

XSI

If OPOST is set, output data is post-processed as described below, so that lines of text are modified to appear appropriately on the terminal device; otherwise, characters are transmitted without change.

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 indicates no delay. If OFILL is set, fill characters are 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 is DEL; otherwise, NUL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

New-line delay lasts about 0.10 seconds. If ONLRET is set, the carriage-return delays are used instead of the newline delays. If OFILL is set, two fill characters are transmitted.

7258 Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10
 7259 seconds, and type 3 is about 0.15 seconds. If OFILL is set, delay type 1 transmits two fill
 7260 characters, and type 2, four fill characters.

7261 Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10
 7262 seconds. Type 3 specifies that tabs shall be expanded into spaces. If OFILL is set, two fill
 7263 characters are transmitted for any delay.

7264 Backspace delay lasts about 0.05 seconds. If OFILL is set, one fill character is transmitted.

7265 The actual delays depend on line speed and system load.

7266 The initial output control value after *open()* is implementation-defined.

7267 11.2.4 Control Modes

7268 The **c_cflag** field describes the hardware control of the terminal, and is composed of the
 7269 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name
 7270 symbols in this table are defined in `<termios.h>`; not all values specified are required to be
 7271 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.

7284 In addition, the input and output baud rates are stored in the **termios** structure. The following
 7285 values are supported:

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

7295 The following functions are provided for getting and setting the values of the input and output
 7296 baud rates in the **termios** structure: *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, and *cfsetospeed()*.
 7297 The effects on the terminal device do not become effective and not all errors are detected until
 7298 the *tcsetattr()* function is successfully called.

7299 The CSIZE bits specify the number of transmitted or received bits per byte. If ISTRIP is not set,
 7300 the value of all the other bits is unspecified. If ISTRIP is set, the value of all but the 7 low-order
 7301 bits is zero, but the value of any other bits beyond CSIZE is unspecified when read. CSIZE does
 7302 not include the parity bit, if any. If CSTOPB is set, two stop bits are used; otherwise, one stop

- 7303 bit. For example, at 110 baud, two stop bits are normally used.
- 7304 If CREAD is set, the receiver is enabled; otherwise, no characters shall be received.
- 7305 If PARENB is set, parity generation and detection is enabled and a parity bit is added to each
7306 byte. If parity is enabled, PARODD specifies odd parity if set; otherwise, even parity is used.
- 7307 If HUPCL is set, the modem control lines for the port are lowered when the last process with the
7308 port open closes the port or the process terminates. The modem connection shall be broken.
- 7309 If CLOCAL is set, a connection does not depend on the state of the modem status lines. If
7310 CLOCAL is clear, the modem status lines shall be monitored.
- 7311 Under normal circumstances, a call to the *open()* function shall wait for the modem connection
7312 to complete. However, if the O_NONBLOCK flag is set (see *open()*) or if CLOCAL has been set,
7313 the *open()* function shall return immediately without waiting for the connection.
- 7314 If the object for which the control modes are set is not an asynchronous serial connection, some
7315 of the modes may be ignored; for example, if an attempt is made to set the baud rate on a
7316 network connection to a terminal on another host, the baud rate may or may not be set on the
7317 connection between that terminal and the machine to which it is directly connected.
- 7318 The initial hardware control value after *open()* is implementation-defined.

7319 11.2.5 Local Modes

- 7320 The *c_lflag* field of the argument structure is used to control various functions. It is composed
7321 of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name
7322 symbols in this table are defined in *<termios.h>*; not all values specified are required to be
7323 supported by the underlying hardware:

7324

7325

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.

7326

7327

7328

7329

7330

7331

7332

7333

7334

- 7335 If ECHO is set, input characters are echoed back to the terminal. If ECHO is clear, input
7336 characters are not echoed.

- 7337 If ECHOE and ICANON are set, the ERASE character shall cause the terminal to erase, if
7338 possible, the last character in the current line from the display. If there were no character to
7339 erase, an implementation might echo an indication that this was the case, or do nothing.

- 7340 If ECHOK and ICANON are set, the KILL character shall either cause the terminal to erase the
7341 line from the display or shall echo the newline character after the KILL character.

- 7342 If ECHONL and ICANON are set, the newline character shall be echoed even if ECHO is not set.

- 7343 If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions,
7344 and the assembly of input characters into lines delimited by NL, EOF, and EOL, as described in
7345 Section 11.1.6 (on page 215).

7346 If ICANON is not set, read requests are satisfied directly from the input queue. A read shall not
 7347 be satisfied until at least MIN bytes have been received or the timeout value TIME expired
 7348 between bytes. The time value represents tenths of a second. See Section 11.1.7 (on page 216) for
 7349 more details.

7350 If IEXTEN is set, implementation-defined functions are recognized from the input data. It is
 7351 implementation-defined how IEXTEN being set interacts with ICANON, ISIG, IXON, or IXOFF.
 7352 If IEXTEN is not set, implementation-defined functions shall not be recognized and the
 7353 corresponding input characters are processed as described for ICANON, ISIG, IXON, and
 7354 IXOFF.

7355 If ISIG is set, each input character is checked against the special control characters INTR, QUIT,
 7356 and SUSP. If an input character matches one of these control characters, the function associated
 7357 with that character is performed. If ISIG is not set, no checking is done. Thus these special input
 7358 functions are possible only if ISIG is set.

7359 If NOFLSH is set, the normal flush of the input and output queues associated with the INTR,
 7360 QUIT, and SUSP characters shall not be done.

7361 If TOSTOP is set, the signal SIGTTOU is sent to the process group of a process that tries to write
 7362 to its controlling terminal if it is not in the foreground process group for that terminal. This
 7363 signal, by default, stops the members of the process group. Otherwise, the output generated by
 7364 that process is output to the current output stream. Processes that are blocking or ignoring
 7365 SIGTTOU signals are excepted and allowed to produce output, and the SIGTTOU signal is not
 7366 sent.

7367 The initial local control value after *open()* is implementation-defined.

7368 11.2.6 Special Control Characters

7369 The special control characters values are defined by the array `c_cc`. The subscript name and
 7370 description for each element in both canonical and non-canonical modes are as follows:

7371

7372

7373

7374

7375

7376

7377

7378

7379

7380

7381

7382

7383

7384

7385

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

7386 The subscript values are unique, except that the VMIN and VTIME subscripts may have the
 7387 same values as the VEOF and VEOL subscripts, respectively.

7388 Implementations that do not support changing the START and STOP characters may ignore the
 7389 character values in the `c_cc` array indexed by the VSTART and VSTOP subscripts when
 7390 *tcsetattr()* is called, but shall return the value in use when *tcgetattr()* is called.

7391 The initial values of all control characters are implementation-defined. |

7392 If the value of one of the changeable special control characters (see Section 11.1.9 (on page 217))
7393 is `_POSIX_VDISABLE`, that function shall be disabled; that is, no input data is recognized as the
7394 disabled special character. If `ICANON` is not set, the value of `_POSIX_VDISABLE` has no special
7395 meaning for the `VMIN` and `VTIME` entries of the `c_cc` array.

7397

7398 **12.1 Utility Argument Syntax**

7399 This section describes the argument syntax of the standard utilities and introduces terminology
 7400 used throughout IEEE Std. 1003.1-200x for describing the arguments processed by the utilities.

7401 Within IEEE Std. 1003.1-200x, a special notation is used for describing the syntax of a utility's
 7402 arguments. Unless otherwise noted, all utility descriptions use this notation, which is illustrated
 7403 by this example (see the Shell and Utilities volume of IEEE Std. 1003.1-200x, Section 2.9.1, Simple
 7404 Commands):

```
7405     utility_name[-a][-b][-c option_argument]
7406             [-d|-e][-foption_argument][operand...]
```

7407 The notation used for the SYNOPSIS sections imposes requirements on the implementors of the
 7408 standard utilities and provides a simple reference for the application developer or system user.

7409 1. The utility in the example is named *utility_name*. It is followed by *options*, *option-*
 7410 *arguments*, and *operands*. The arguments that consist of hyphens and single letters or
 7411 digits, such as 'a', are known as *options* (or, historically, *flags*). Certain options are
 7412 followed by an *option-argument*, as shown with [-c *option_argument*]. The arguments
 7413 following the last options and option-arguments are named *operands*.

7414 2. Option-arguments are sometimes shown separated from their options by <blank>
 7415 characters, sometimes directly adjacent. This reflects the situation that in some cases an
 7416 option-argument is included within the same argument string as the option; in most cases
 7417 it is the next argument. The Utility Syntax Guidelines in Section 12.2 (on page 229) require
 7418 that the option be a separate argument from its option-argument, but there are some
 7419 exceptions in IEEE Std. 1003.1-200x to ensure continued operation of historical
 7420 applications:

7421 a. If the SYNOPSIS of a standard utility shows a space character between an option and
 7422 option-argument (as with [-c *option_argument*] in the example), a portable
 7423 application shall use separate arguments for that option and its option-argument.

7424 b. If a space character is not shown (as with [-*foption_argument*] in the example), a
 7425 portable application shall place an option and its option-argument directly adjacent
 7426 in the same argument string, without intervening <blank> characters.

7427 c. Notwithstanding the preceding requirements on portable applications, a conforming
 7428 system shall permit, but shall not require, an application to specify options and
 7429 option-arguments as separate arguments whether or not a space character is shown
 7430 XSI on the synopsis line, except in those cases (marked with the XSI portability warning)
 7431 where an option-argument is optional and no separation can be used.

7432 d. A standard utility may also be implemented to operate correctly when the required
 7433 separation into multiple arguments is violated by a non-portable application.

7434 In summary, the following table shows allowable combinations:

7435
7436
7437
7438
7439
7440
7441
7442
7443
7444
7445
7446
7447
7448
7449
7450
7451
7452
7453
7454
7455
7456
7457
7458
7459
7460
7461
7462
7463
7464
7465
7466
7467
7468
7469
7470
7471
7472
7473
7474
7475
7476
7477
7478

	SYNOPSIS Shows:		
	-a arg	-barg	-c[arg]
Portable application shall use:	-a arg	-barg	N/A
System shall support:	-a arg	-barg	-carg or -c
System may support:	-aarg	-b arg	

3. 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 229) applies. If an option that does not have option-arguments is repeated, the results are undefined, unless otherwise stated.
4. Frequently, names of parameters that require substitution by actual values are shown with embedded underscores. Alternatively, parameters are shown as follows:

<parameter name>

The angle brackets are used for the symbolic grouping of a phrase representing a single parameter and portable applications shall not include them in data submitted to the utility.

5. 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*]*

Utilities with very complex arguments may be shown as follows:

*utility_name [options][*operands*]*

6. Unless otherwise specified, whenever an operand or option-argument is, or contains, a numeric value:
 - The number is interpreted as a decimal integer.
 - Numerals in the range 0 to 2 147 483 647 are syntactically recognized as numeric values.
 - 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.
 - Ranges greater than those listed here are allowed.

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.

7. Arguments or option-arguments enclosed in the '[' and ']' notation are optional and can be omitted. Portable applications shall not include the '[' and ']' symbols in data submitted to the utility.
8. Arguments separated by the '|' vertical bar notation are mutually-exclusive. Portable applications shall not include the '|' symbol in data submitted to the utility. Alternatively, mutually-exclusive options and operands may be listed with multiple

7479 synopsis lines. For example:

```
7480     utility_name -d[-a][[-c option_argument]][operand...]
7481     utility_name[-a][[-b]][operand...]
```

7482 When multiple synopsis lines are given for a utility, it is an indication that the utility has
 7483 mutually-exclusive arguments. These mutually-exclusive arguments alter the functionality
 7484 of the utility so that only certain other arguments are valid in combination with one of the
 7485 mutually-exclusive arguments. Only one of the mutually-exclusive arguments is allowed
 7486 for invocation of the utility. Unless otherwise stated in an accompanying OPTIONS
 7487 section, the relationships between arguments depicted in the SYNOPSIS sections are
 7488 mandatory requirements placed on portable applications. The use of conflicting mutually-
 7489 exclusive arguments produces undefined results, unless a utility description specifies
 7490 otherwise. When an option is shown without the '[' and ']' brackets, it means that
 7491 option is required for that version of the SYNOPSIS. However, it is not required to be the
 7492 first argument, as shown in the example above, unless otherwise stated.

7493 9. Ellipses ("...") are used to denote that one or more occurrences of an option or operand
 7494 are allowed. When an option or an operand followed by ellipses is enclosed in brackets,
 7495 zero or more options or operands can be specified. The forms:

```
7496     utility_name -f option_argument...[operand...]  

7497     utility_name [-g option_argument]...[operand...]
```

7498 indicate that multiple occurrences of the option and its option-argument preceding the
 7499 ellipses are valid, with semantics as indicated in the OPTIONS section of the utility. (See
 7500 also Guideline 11 in Section 12.2.) In the first example, each option-argument requires a
 7501 preceding `-f` and at least one `-f option_argument` must be given.

7502 10. When the synopsis line is too long to be printed on a single line in the Shell and Utilities
 7503 volume of IEEE Std. 1003.1-200x, the indented lines following the initial line are
 7504 continuation lines. An actual use of the command would appear on a single logical line.

7505 12.2 Utility Syntax Guidelines

7506 The following guidelines are established for the naming of utilities and for the specification of
 7507 options, option-arguments, and operands. The `getopt()` function in the System Interfaces volume
 7508 of IEEE Std. 1003.1-200x assists utilities in handling options and operands that conform to these
 7509 guidelines.

7510 Operands and option-arguments can contain characters not specified in the portable character
 7511 set.

7512 The guidelines are intended to provide guidance to the authors of future utilities, such as those
 7513 written specific to a local system or that are components of a larger application. Some of the
 7514 standard utilities do not conform to all of these guidelines; in those cases, the OPTIONS sections
 7515 describe the deviations.

7516 **Guideline 1:** Utility names should be between two and nine characters, inclusive.

7517 **Guideline 2:** Utility names should include lowercase letters (the **lower** character
 7518 classification) and digits only from the portable character set.

7519 **Guideline 3:** Each option name should be a single alphanumeric character (the **alnum**
 7520 character classification) from the portable character set.

- 7521 Multi-digit options are not allowed.
- 7522 **Guideline 4:** All options should be preceded by the ‘-’ delimiter character.
- 7523 **Guideline 5:** Options without option-arguments should be accepted when grouped behind
7524 one ‘-’ delimiter.
- 7525 **Guideline 6:** Each option and option-argument should be a separate argument, except as
7526 noted in Section 12.1 (on page 227), item (2).
- 7527 **Guideline 7:** Option-arguments should not be optional.
- 7528 **Guideline 8:** When multiple option-arguments are specified to follow a single option, they
7529 should be presented as a single argument, using commas within that
7530 argument or <blank> characters within that argument to separate them.
- 7531 **Guideline 9:** All options should precede operands on the command line.
- 7532 **Guideline 10:** The argument -- should be accepted as a delimiter indicating the end of
7533 options. Any following arguments should be treated as operands, even if they
7534 begin with the ‘-’ character. The -- argument should not be used as an
7535 option or as an operand.
- 7536 **Guideline 11:** The order of different options relative to one another should not matter,
7537 unless the options are documented as mutually-exclusive and such an option
7538 is documented to override any incompatible options preceding it. If an option
7539 that has option-arguments is repeated, the option and option-argument
7540 combinations should be interpreted in the order specified on the command
7541 line.
- 7542 **Guideline 12:** The order of operands may matter and position-related interpretations should
7543 be determined on a utility-specific basis.
- 7544 **Guideline 13:** For utilities that use operands to represent files to be opened for either reading
7545 or writing, the ‘-’ operand should be used only to mean standard input (or
7546 standard output when it is clear from context that an output file is being
7547 specified).
- 7548 The utilities in the Shell and Utilities volume of IEEE Std. 1003.1-200x that claim conformance to
7549 these guidelines shall conform completely to these guidelines as if these guidelines contained the
7550 term “shall” instead of “should”. On some systems, the utilities accept usage in violation of
7551 these guidelines for backward compatibility as well as accepting the required form.
- 7552 It is recommended that all future utilities and applications use these guidelines to enhance user
7553 portability. The fact that some historical utilities could not be changed (to avoid breaking
7554 existing applications) should not deter this future goal.

7555

7556 This chapter describes the contents of headers.

7557 Headers contain function prototypes, the definition of symbolic constants, common structures,
7558 preprocessor macros, and defined types. Each function in the System Interfaces volume of
7559 IEEE Std. 1003.1-200x specifies the headers that an application shall include in order to use that
7560 function. In most cases, only one header is required. These headers are present on an application
7561 development system; they need not be present on the target execution system.

7562 **13.1 Format of Entries**

7563 The entries in this chapter are based on a common format as follows. The only sections relating
7564 to conformance are the SYNOPSIS and DESCRIPTION.

7565 **NAME**

7566 This section gives the name or names of the entry and briefly states its purpose.

7567 **SYNOPSIS**

7568 This section summarizes the use of the entry being described.

7569 **DESCRIPTION**

7570 This section describes the functionality of the header.

7571 **APPLICATION USAGE**

7572 This section is non-normative.

7573 This section gives warnings and advice to application writers about the entry. In the
7574 event of conflict between warnings and advice and a normative part of this volume of
7575 IEEE Std. 1003.1-200x, the normative material is to be taken as correct.

7576 **RATIONALE**

7577 This section is non-normative.

7578 This section contains historical information concerning the contents of this volume of
7579 IEEE Std. 1003.1-200x and why features were included or discarded by the standard
7580 developers.

7581 **FUTURE DIRECTIONS**

7582 This section is non-normative.

7583 This section provides comments which should be used as a guide to current thinking;
7584 there is not necessarily a commitment to adopt these future directions.

7585 **SEE ALSO**

7586 This section is non-normative.

7587 This section gives references to related information.

7588 **CHANGE HISTORY**

7589 This section is non-normative.

7590 This section shows the derivation of the entry and any significant changes that have
7591 been made to it.

7592 **NAME**7593 aio.h — asynchronous input and output (**REALTIME**)7594 **SYNOPSIS**

7595 AIO #include <aio.h>

7596

7597 **DESCRIPTION**7598 The <aio.h> header shall define the **aio** structure which shall include at least the following
7599 members:

7600	int	aio_fildes	File descriptor.
7601	off_t	aio_offset	File offset.
7602	volatile void *	aio_buf	Location of buffer.
7603	size_t	aio_nbytes	Length of transfer.
7604	int	aio_reqprio	Request priority offset.
7605	struct sigevent	aio_sigevent	Signal number and value.
7606	int	aio_lio_opcode	Operation to be performed.

7607 This header shall also include the following constants:

7608 **AIO_CANCELED** A return value indicating that all requested operations have been
7609 canceled.7610 **AIO_NOTCANCELED**7611 A return value indicating that some of the requested operations could not
7612 be canceled since they are in progress.7613 **AIO_ALLDONE** A return value indicating that none of the requested operations could be
7614 canceled since they are already complete.7615 **LIO_WAIT** A *lio_listio()* synchronization operation indicating that the calling thread
7616 is to suspend until the *lio_listio()* operation is complete.7617 **LIO_NOWAIT** A *lio_listio()* synchronization operation indicating that the calling thread
7618 is to continue execution while the *lio_listio()* operation is being
7619 performed, and no notification is given when the operation is complete.7620 **LIO_READ** A *lio_listio()* element operation option requesting a read.7621 **LIO_WRITE** A *lio_listio()* element operation option requesting a write.7622 **LIO_NOP** A *lio_listio()* element operation option indicating that no transfer is
7623 requested.7624 The following shall be declared as functions and may also be declared as macros. Function
7625 prototypes shall be provided for use with an ISO C standard compiler.

```

7626 int aio_cancel(int, struct aiocb *);
7627 int aio_error(const struct aiocb *);
7628 int aio_fsync(int, struct aiocb *);
7629 int aio_read(struct aiocb *);
7630 ssize_t aio_return(struct aiocb *);
7631 int aio_suspend(const struct aiocb *const[], int,
7632               const struct timespec *);
7633 int aio_write(struct aiocb *);
7634 int lio_listio(int, struct aiocb *restrict const[restrict], int,
7635               struct sigevent *restrict);

```


7636 Inclusion of the <aio.h> header may make visible symbols defined in the headers <fcntl.h>,
7637 <signal.h>, <sys/types.h>, and <time.h>.

7638 **APPLICATION USAGE**

7639 None.

7640 **RATIONALE**

7641 None.

7642 **FUTURE DIRECTIONS**

7643 None.

7644 **SEE ALSO**

7645 <fcntl.h>, <signal.h>, <sys/types.h>, <time.h>, the System Interfaces volume of
7646 IEEE Std. 1003.1-200x, *fsync()*, *lseek()*, *read()*, *write()*

7647 **CHANGE HISTORY**

7648 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

7649 **Issue 6**

7650 The <aio.h> header is marked as part of the Asynchronous Input and Output option. |

7651 The description of the constants is expanded. |

7652 The **restrict** keyword is added to the prototype for *lio_listio()*. |

7653 **NAME**

7654 arpa/inet.h — definitions for internet operations

7655 **SYNOPSIS**

7656 #include <arpa/inet.h>

7657 **DESCRIPTION**7658 The **in_port_t** and **in_addr_t** types shall be defined as described in <netinet/in.h>.7659 The **in_addr** structure shall be defined as described in <netinet/in.h>.7660 The INET_ADDRSTRLEN and INET6_ADDRSTRLEN macros shall be defined as described in
7661 <netinet/in.h>.7662 The following shall be declared as functions, defined as macros, or both. If functions are
7663 declared, function prototypes shall be provided for use with an ISO C standard compiler.

7664 uint32_t htonl(uint32_t);

7665 uint16_t htons(uint16_t);

7666 uint32_t ntohl(uint32_t);

7667 uint16_t ntohs(uint16_t);

7668 The **uint32_t** and **uint16_t** types shall be defined as described in <inttypes.h>.7669 The following shall be declared as functions, and may also be defined as macros. Function
7670 prototypes shall be provided for use with an ISO C standard compiler.

7671 in_addr_t inet_addr(const char *);

7672 in_addr_t inet_lnaof(struct in_addr);

7673 struct in_addr inet_makeaddr(in_addr_t, in_addr_t);

7674 in_addr_t inet_netof(struct in_addr);

7675 in_addr_t inet_network(const char *);

7676 char *inet_ntoa(struct in_addr);

7677 IP6 const char *inet_ntop(int, const void *restrict, char *restrict,
7678 socklen_t);

7679 int inet_pton(int, const char *restrict, void *restrict);

7680

7681 Inclusion of the <arpa/inet.h> header may also make visible all symbols from <netinet/in.h>
7682 and <inttypes.h>.7683 **APPLICATION USAGE**

7684 None.

7685 **RATIONALE**

7686 None.

7687 **FUTURE DIRECTIONS**

7688 None.

7689 **SEE ALSO**7690 <netinet/in.h>, <inttypes.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *htonl()*,
7691 *inet_addr()*7692 **CHANGE HISTORY**

7693 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

7694 The **restrict** keyword is added to the prototypes for *inet_ntop()* and *inet_pton()*.

7695 **NAME**

7696 assert.h — verify program assertion

7697 **SYNOPSIS**

7698 #include <assert.h>

7699 **DESCRIPTION**

7700 **CX** The functionality described on this reference page extends the ISO C standard. Applications
7701 shall define the appropriate feature test macro (see the System Interfaces volume of
7702 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
7703 symbols in this header.

7704 The <assert.h> header shall define the *assert()* macro. It refers to the macro *NDEBUG* which is
7705 not defined in the header. If *NDEBUG* is defined as a macro name before the inclusion of this
7706 header, the *assert()* macro is defined simply as:

7707

```
#define assert(ignore)((void) 0)
```

7708 Otherwise, the macro behaves as described in *assert()*.

7709 The *assert()* macro is redefined according to the current state of *NDEBUG* each time <assert.h>
7710 is included.

7711 The *assert()* macro is implemented as a macro, not as a function. If the macro definition is
7712 suppressed in order to access an actual function, the behavior is undefined.

7713 **APPLICATION USAGE**

7714 None.

7715 **RATIONALE**

7716 None.

7717 **FUTURE DIRECTIONS**

7718 None.

7719 **SEE ALSO**7720 The System Interfaces volume of IEEE Std. 1003.1-200x, *assert()*7721 **CHANGE HISTORY**

7722 First released in Issue 1. Derived from Issue 1 of the SVID.

7723 **Issue 6**

7724 The definition of the *assert()* macro is changed for alignment with the ISO/IEC 9899:1999
7725 standard.

7726 **NAME**7727 `complex.h` — complex arithmetic7728 **SYNOPSIS**7729 `#include <complex.h>`7730 **DESCRIPTION**

7731 `CX` The functionality described on this reference page extends the ISO C standard. Applications
 7732 shall define the appropriate feature test macro (see the System Interfaces volume of
 7733 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 7734 symbols in this header.

7735 The **<complex.h>** header shall define the following constants:

7736 `complex` Expands to `_Complex`.

7737 `_Complex_I` Expands to a constant expression of type `const float _Complex`, with the value
 7738 of the imaginary unit (that is, a number such that $i^2 = -1$).

7739 `imaginary` Expands to `_Imaginary`.

7740 `_Imaginary_I` Expands to a constant expression of type `const float _Imaginary` with the value
 7741 of the imaginary unit.

7742 `I` Expands to either `_Imaginary_I` or `_Complex_I`. If `_Imaginary_I` is not defined, `I`
 7743 expands to `_Complex_I`.

7744 The constants `imaginary` and `_Imaginary_I` shall be defined if the implementation supports
 7745 imaginary types.

7746 An application may undefine and then, perhaps, redefine the `complex`, `imaginary`, and `I` constants.

7747 The following shall be declared as functions and may also be defined as macros. Function
 7748 prototypes shall be provided for use with an ISO C standard compiler.

```

7749 double      complex cacos(double complex);
7750 float       complex cacosf(float complex);
7751 long double complex cacosl(long double complex);
7752 double      complex casin(double complex);
7753 float       complex casinf(float complex);
7754 long double complex casinl(long double complex);
7755 double      complex catan(double complex);
7756 float       complex catanf(float complex);
7757 long double complex catanl(long double complex);
7758 double      complex ccos(double complex);
7759 float       complex ccosf(float complex);
7760 long double complex ccosl(long double complex);
7761 double      complex csin(double complex);
7762 float       complex csinf(float complex);
7763 long double complex csinl(long double complex);
7764 double      complex ctan(double complex);
7765 float       complex ctanf(float complex);
7766 long double complex ctanl(long double complex);
7767 double      complex cacosh(double complex);
7768 float       complex cacoshf(float complex);
7769 long double complex cacoshl(long double complex);
7770 double      complex casinh(double complex);
7771 float       complex casinhf(float complex);

```

```
7772     long double  complex casinhl(long double complex);
7773     double       complex catanh(double complex);
7774     float        complex catanhf(float complex);
7775     long double  complex catanhl(long double complex);
7776     double       complex ccosh(double complex);
7777     float        complex ccoshf(float complex);
7778     long double  complex ccoshl(long double complex);
7779     double       complex csinh(double complex);
7780     float        complex csinhf(float complex);
7781     long double  complex csinhl(long double complex);
7782     double       complex catanh(double complex);
7783     float        complex catanhf(float complex);
7784     long double  complex catanhl(long double complex);
7785     double       complex cexp(double complex);
7786     float        complex cexpf(float complex);
7787     long double  complex cexpl(long double complex);
7788     double       complex clog(double complex);
7789     float        complex clogf(float complex);
7790     long double  complex clogl(long double complex);
7791     double       complex cabs(double complex);
7792     float        complex cabsf(float complex);
7793     long double  complex cabsl(long double complex);
7794     double       complex cpow(double complex, double complex);
7795     float        complex cpowf(float complex, float complex);
7796     long double  complex cpowl(long double complex, long double complex);
7797     double       complex csqrt(double complex);
7798     float        complex csqrtf(float complex);
7799     long double  complex csqrtl(long double complex);
7800     double       complex carg(double complex);
7801     float        complex cargf(float complex);
7802     long double  complex cargl(long double complex);
7803     double       complex cimag(double complex);
7804     float        complex cimagf(float complex);
7805     long double  complex cimagl(long double complex);
7806     double       complex conj(double complex);
7807     float        complex conjf(float complex);
7808     long double  complex conjl(long double complex);
7809     double       complex cproj(double complex);
7810     float        complex cprojf(float complex);
7811     long double  complex cprojl(long double complex);
7812     double       complex creal(double complex);
7813     float        complex crealf(float complex);
7814     long double  complex creall(long double complex);
```

7815 **APPLICATION USAGE**

7816 Values are interpreted as radians, not degrees. An implementation may set *errno*, but is not
7817 required to.

7818 Some of the complex arithmetic functions have branch cuts, across which the function is
7819 discontinuous. For implementations with a signed zero (including all IEC 60559:1989 standard
7820 implementations), the sign of zero distinguishes one side of a cut from another so the function is
7821 continuous (except for format limitations) as the cut is approached from either side. For
7822 example, for the square root function, which has a branch cut along the negative real axis, the
7823 top of the cut, with imaginary part +0, maps to the positive imaginary axis, and the bottom of
7824 the cut, with imaginary part -0, maps to the negative imaginary axis.

7825 Implementations that do not support a signed zero cannot distinguish the sides of branch cuts.
7826 These implementations shall map a cut so the function is continuous as the cut is approached
7827 coming around the finite endpoint of the cut in a counter-clockwise direction. (Branch cuts for
7828 the functions specified here have just one finite endpoint.) For example, for the square root
7829 function, coming counter-clockwise around the finite endpoint of the cut along the negative real
7830 axis approaches the cut from above, so the cut maps to the positive imaginary axis.

7831 The usual mathematical formulas for complex multiply, divide, and absolute value are
7832 problematic because of their treatment of infinities and because of undue overflow and
7833 underflow. The `CX_LIMITED_RANGE` pragma can be used to inform the implementation that
7834 (where the state is on) the usual mathematical formulas are acceptable. The pragma can occur
7835 either outside external declarations or preceding all explicit declarations and statements inside a
7836 compound statement. When outside external declarations, the pragma takes effect from its
7837 occurrence until another `CX_LIMITED_RANGE` pragma is encountered, or until the end of the
7838 translation unit. When inside a compound statement, the pragma takes effect from its
7839 occurrence until another `CX_LIMITED_RANGE` pragma is encountered (including within a
7840 nested compound statement), or until the end of the compound statement; at the end of a
7841 compound statement the state for the pragma is restored to its condition just before the
7842 compound statement. If this pragma is used in any other context, the behavior is undefined. The
7843 default state for the pragma is off.

7844 **RATIONALE**

7845 The choice of *I* instead of *i* for the imaginary unit concedes to the widespread use of the
7846 identifier *i* for other purposes. The application can use a different identifier, say *j*, for the
7847 imaginary unit by following the inclusion of the `<complex.h>` header with:

```
7848 #undef I  
7849 #define j _Imaginary_I
```

7850 An *I* suffix to designate imaginary constants is not required, as multiplication by *I* provides a
7851 sufficiently convenient and more generally useful notation for imaginary terms. The
7852 corresponding real type for the imaginary unit is `float`, so that use of *I* for algorithmic or
7853 notational convenience will not result in widening types.

7854 On systems with imaginary types, the application has the ability to control whether use of the
7855 macro *I* introduces an imaginary type, by explicitly defining *I* to be `_Imaginary_I` or `_Complex_I`.
7856 Disallowing imaginary types is useful for some applications intended to run on implementations
7857 without support for such types.

7858 The macro `_Imaginary_I` provides a test for whether imaginary types are supported.

7859 The `cis()` function ($\cos(x) + I^*\sin(x)$) was considered but rejected because its implementation is
7860 easy and straightforward, even though some implementations could compute sine and cosine
7861 more efficiently in tandem.

7862 **FUTURE DIRECTIONS**

7863 The following function names and the same names suffixed with *f* or *l* are reserved for future
7864 use, and may be added to the declarations in the <complex.h> header.

7865	<i>cerf()</i>	<i>cexpm1()</i>	<i>clog2()</i>
7866	<i>cerfc()</i>	<i>clog10()</i>	<i>clgamma()</i>
7867	<i>cexp2()</i>	<i>clog1p()</i>	<i>ctgamma()</i>

7868 **SEE ALSO**

7869 The System Interfaces volume of IEEE Std. 1003.1-200x, *cabs()*, *cacos()*, *cacosh()*, *carg()*, *casin()*,
7870 *casinh()*, *catan()*, *catanh()*, *ccos()*, *ccosh()*, *cexp()*, *cimag()*, *clog()*, *conj()*, *cpow()*, *cproj()*, *creal()*,
7871 *csin()*, *csinh()*, *csqrt()*, *ctan()*, *ctanh()*

7872 **CHANGE HISTORY**

7873 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

7874 **NAME**

7875 cpio.h — cpio archive values

7876 **SYNOPSIS**

7877 xSI #include <cpio.h>

7878

7879 **DESCRIPTION**

7880 Values needed by the *c_mode* field of the *cpio* archive format are described as follows:

7881

7882

	Name	Description	Value (Octal)
7883	C_IRUSR	Read by owner.	0000400
7884	C_IWUSR	Write by owner.	0000200
7885	C_IXUSR	Execute by owner.	0000100
7886	C_IRGRP	Read by group.	0000040
7887	C_IWGRP	Write by group.	0000020
7888	C_IXGRP	Execute by group.	0000010
7889	C_IROTH	Read by others.	0000004
7890	C_IWOTH	Write by others.	0000002
7891	C_IXOTH	Execute by others.	0000001
7892	C_ISUID	Set user ID.	0004000
7893	C_ISGID	Set group ID.	0002000
7894	C_ISVTX	On directories, restricted deletion flag.	0001000
7895	C_ISDIR	Directory.	0040000
7896	C_ISFIFO	FIFO.	0010000
7897	C_ISREG	Regular file.	0100000
7898	C_ISBLK	Block special.	0060000
7899	C_ISCHR	Character special.	0020000
7900	C_ISCTG	Reserved.	0110000
7901	C_ISLNK	Symbolic link.	0120000
7902	C_ISSOCK	Socket.	0140000

7903 The header shall define the symbolic constant:

7904 MAGIC "070707"

7905 **APPLICATION USAGE**

7906 None.

7907 **RATIONALE**

7908 None.

7909 **FUTURE DIRECTIONS**

7910 None.

7911 **SEE ALSO**

7912 The Shell and Utilities volume of IEEE Std. 1003.1-200x, *pax*

7913 **CHANGE HISTORY**

7914 First released in Issue 3 of the Headers Interface, Issue 3 specification. Derived from the

7915 POSIX.1-1988 standard.

7916 **Issue 4, Version 2**

7917 Descriptions for C_ISLNK and C_ISSOCK are provided; formerly, these were listed as

7918 "Reserved".

7919 **Issue 6**

7920

7921

The SEE ALSO is updated to refer to *pax*, since the *cpio* utility is not included in the Shell and Utilities volume of IEEE Std. 1003.1-200x.

7922 **NAME**

7923 ctype.h — character types

7924 **SYNOPSIS**

7925 #include <ctype.h>

7926 **DESCRIPTION**

7927 **CX** The functionality described on this reference page extends the ISO C standard. Applications
7928 shall define the appropriate feature test macro (see the System Interfaces volume of
7929 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
7930 symbols in this header.

7931 The **<ctype.h>** header shall declare the following as functions and may also define them as
7932 macros. Function prototypes shall be provided for use with an ISO C standard compiler.

7933 int isalnum(int);

7934 int isalpha(int);

7935 **XSI** int isascii(int);

7936 int isblank(int);

7937 int iscntrl(int);

7938 int isdigit(int);

7939 int isgraph(int);

7940 int islower(int);

7941 int isprint(int);

7942 int ispunct(int);

7943 int isspace(int);

7944 int isupper(int);

7945 int isxdigit(int);

7946 **XSI** int toascii(int);

7947 int tolower(int);

7948 int toupper(int);

7949 The following are defined as macros:

7950 **XSI** int _toupper(int);

7951 int _tolower(int);

7952

7953 **APPLICATION USAGE**

7954 None.

7955 **RATIONALE**

7956 None.

7957 **FUTURE DIRECTIONS**

7958 None.

7959 **SEE ALSO**

7960 **<locale.h>**, the System Interfaces volume of IEEE Std. 1003.1-200x, *isalnum()*, *isalpha()*, *isascii()*,
7961 *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *mblen()*,
7962 *mbstowcs()*, *mbtowc()*, *setlocale()*, *toascii()*, *tolower()*, *_tolower()*, *toupper()*, *_toupper()*, *wcstombs()*,
7963 *wctomb()*

7964 **CHANGE HISTORY**

7965 First released in Issue 1. Derived from Issue 1 of the SVID.

7966 **Issue 4**

7967 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 7968
 - The function declarations in this header are expanded to full ISO C standard prototypes.

7969 **Issue 6**

7970 Extensions beyond the ISO C standard are now marked.

7971 **NAME**7972 `dirent.h` — format of directory entries7973 **SYNOPSIS**7974 `#include <dirent.h>`7975 **DESCRIPTION**

7976 The internal format of directories is unspecified.

7977 The **<dirent.h>** header shall define the following data type through **typedef**:7978 **DIR** A type representing a directory stream.7979 It shall also define the structure **dirent** which shall include the following members:7980 XSI `ino_t d_ino` File serial number.7981 `char d_name[]` Name of entry.7982 XSI The type **ino_t** shall be defined as described in **<sys/types.h>**.7983 The character array `d_name` is of unspecified size, but the number of bytes preceding the
7984 terminating null byte does not exceed `{NAME_MAX}`.7985 The following shall be declared as functions and may also be defined as macros. Function
7986 prototypes shall be provided for use with an ISO C standard compiler.7987 `int closedir(DIR *);`7988 `DIR *opendir(const char *);`7989 `struct dirent *readdir(DIR *);`7990 TSF `int readdir_r(DIR *restrict, struct dirent *restrict,
7991 struct dirent **restrict);`7992 `void rewinddir(DIR *);`7993 XSI `void seekdir(DIR *, long);`7994 `long telldir(DIR *);`

7995

7996 **APPLICATION USAGE**

7997 None.

7998 **RATIONALE**7999 Information similar to that in the **<dirent.h>** header is contained in a file **<sys/dir.h>** in 4.2 BSD
8000 and 4.3 BSD. The equivalent in these implementations of **struct dirent** from this volume of
8001 IEEE Std. 1003.1-200x is **struct direct**. The file name was changed because the name **<sys/dir.h>**
8002 was also used in earlier implementations to refer to definitions related to the older access
8003 method; this produced name conflicts. The name of the structure was changed because this
8004 volume of IEEE Std. 1003.1-200x does not completely define what is in the structure, so it could
8005 be different on some implementations from **struct direct**.8006 The name of an array of **char** of an unspecified size should not be used as an **lvalue**. Use of:8007 `sizeof(d_name)`

8008 is incorrect; use:

8009 `strlen(d_name)`

8010 instead.

8011 The array of **char** `d_name` is not a fixed size. Implementations may need to declare **struct dirent**
8012 with an array size for `d_name` of 1, but the actual number of characters provided matches (or
8013 only slightly exceeds) the length of the file name.

8014 **FUTURE DIRECTIONS**

8015 None.

8016 **SEE ALSO**

8017 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *closedir()*, *opendir()*,
8018 *readdir()*, *readdir_r()*, *rewinddir()*, *seekdir()*, *telldir()*

8019 **CHANGE HISTORY**

8020 First released in Issue 2.

8021 **Issue 4**

8022 Reference to type **ino_t** is marked as an extension, as are references to the *seekdir()* and *telldir()*
8023 functions.

8024 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 8025 • The function declarations in this header are expanded to full ISO C standard prototypes.
- 8026 • A statement is added to the DESCRIPTION indicating that the internal format of directories
8027 is unspecified. Also in the description of the *d_name* field, the text is changed to indicate
8028 “bytes” rather than (possibly multi-byte) “characters”.

8029 **Issue 5**

8030 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

8031 **Issue 6**

8032 The Open Group corrigenda item U026/7 has been applied, correcting the prototype for
8033 *readdir_r()*.

8034 The **restrict** keyword is added to the prototype for *readdir_r()*.

8035 **NAME**8036 `dlfcn.h` — dynamic linking8037 **SYNOPSIS**8038 XSI `#include <dlfcn.h>`

8039

8040 **DESCRIPTION**8041 The `<dlfcn.h>` header shall define at least the following macros for use in the construction of a
8042 `dlopen()` *mode* argument:8043 `RTLD_LAZY` Relocations are performed at an implementation-defined time.8044 `RTLD_NOW` Relocations are performed when the object is loaded.8045 `RTLD_GLOBAL` All symbols are available for relocation processing of other modules.8046 `RTLD_LOCAL` All symbols are not made available for relocation processing by other
8047 modules.8048 The `<dlfcn.h>` header shall declare the following functions which may also be defined as
8049 macros. Function prototypes shall be provided for use with an ISO C standard compiler.8050 `int dlclose(void *);`8051 `char *dlerror(void);`8052 `void *dlopen(const char *, int);`8053 `void *dlsym(void *restrict, const char *restrict);`8054 **APPLICATION USAGE**

8055 None.

8056 **RATIONALE**

8057 None.

8058 **FUTURE DIRECTIONS**

8059 None.

8060 **SEE ALSO**8061 The System Interfaces volume of IEEE Std. 1003.1-200x, `dlopen()`, `dlclose()`, `dlsym()`, `dlerror()`8062 **CHANGE HISTORY**

8063 First released in Issue 5.

8064 **Issue 6**8065 The `restrict` keyword is added to the prototype for `dlsym()`.

8066 **NAME**

8067 errno.h — system error numbers

8068 **SYNOPSIS**

8069 #include <errno.h>

8070 **DESCRIPTION**

8071 **cx** The functionality described on this reference page extends the ISO C standard. Applications
 8072 shall define the appropriate feature test macro (see the System Interfaces volume of
 8073 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 8074 symbols in this header.

8075 The <errno.h> header provides a declaration for *errno* and gives non-zero values for the
 8076 following symbolic constants. Their values are unique except as noted below:

8077	[E2BIG]	Argument list too long.
8078	[EACCES]	Permission denied.
8079	[EADDRINUSE]	Address in use.
8080	[EADDRNOTAVAIL]	Address not available.
8081	[EAFNOSUPPORT]	Address family not supported.
8082	[EAGAIN]	Resource unavailable, try again (may be the same value as
8083		[EWOULDBLOCK]).
8084	[EALREADY]	Connection already in progress.
8085	[EBADF]	Bad file descriptor.
8086	[EBADMSG]	Bad message.
8087	[EBUSY]	Device or resource busy.
8088	[ECANCELED]	Operation canceled.
8089	[ECHILD]	No child processes.
8090	[ECONNABORTED]	Connection aborted.
8091	[ECONNREFUSED]	Connection refused.
8092	[ECONNRESET]	Connection reset.
8093	[EDEADLK]	Resource deadlock would occur.
8094	[EDESTADDRREQ]	Destination address required.
8095	[EDOM]	Mathematics argument out of domain of function.
8096	[EDQUOT]	Reserved.
8097	[EEXIST]	File exists.
8098	[EFAULT]	Bad address.
8099	[EFBIG]	File too large.
8100	[EHOSTUNREACH]	Host is unreachable.
8101	[EIDRM]	Identifier removed.
8102	[EILSEQ]	Illegal byte sequence.

8103	[EINPROGRESS]	Operation in progress.	
8104	[EINTR]	Interrupted function.	
8105	[EINVAL]	Invalid argument.	
8106	[EIO]	I/O error.	
8107	[EISCONN]	Socket is connected.	
8108	[EISDIR]	Is a directory.	
8109	[ELOOP]	Too many levels of symbolic links.	
8110	[EMFILE]	Too many open files.	
8111	[EMLINK]	Too many links.	
8112	[EMSGSIZE]	Message too large.	
8113	[EMULTIHOP]	Reserved.	
8114	[ENAMETOOLONG]	File name too long.	
8115	[ENETDOWN]	Network is down.	
8116	[ENETUNREACH]	Network unreachable.	
8117	[ENFILE]	Too many files open in system.	
8118	[ENOBUFS]	No buffer space available.	
8119 XSI	[ENODATA]	No message is available on the STREAM head read queue.	
8120	[ENODEV]	No such device.	
8121	[ENOENT]	No such file or directory.	
8122	[ENOEXEC]	Executable file format error.	
8123	[ENOLCK]	No locks available.	
8124	[ENOLINK]	Reserved.	
8125	[ENOMEM]	Not enough space.	
8126	[ENOMSG]	No message of the desired type.	
8127	[ENOPROTOPT]	Protocol not available.	
8128	[ENOSPC]	No space left on device.	
8129 XSI	[ENOSR]	No STREAM resources.	
8130 XSI	[ENOSTR]	Not a STREAM.	
8131	[ENOSYS]	Function not supported.	
8132	[ENOTCONN]	The socket is not connected.	
8133	[ENOTDIR]	Not a directory.	
8134	[ENOTEMPTY]	Directory not empty.	
8135	[ENOTSOCK]	Not a socket.	
8136	[ENOTSUP]	Not supported.	

8137	[ENOTTY]	Inappropriate I/O control operation.
8138	[ENXIO]	No such device or address.
8139	[EOPNOTSUPP]	Operation not supported on socket.
8140	[EOVERFLOW]	Value too large to be stored in data type.
8141	[EPERM]	Operation not permitted.
8142	[EPIPE]	Broken pipe.
8143	[EPROTO]	Protocol error.
8144	[EPROTONOSUPPORT]	
8145		Protocol not supported.
8146	[EPROTOTYPE]	Socket type not supported.
8147	[ERANGE]	Result too large.
8148	[EROFS]	Read-only file system.
8149	[ESPIPE]	Invalid seek.
8150	[ESRCH]	No such process.
8151	[ESTALE]	Reserved.
8152	<small>XS1</small> [ETIME]	Stream <i>ioctl()</i> timeout.
8153	[ETIMEDOUT]	Connection timed out.
8154	[ETXTBSY]	Text file busy.
8155	[EWOULDBLOCK]	Operation would block (may be the same value as [EAGAIN]).
8156	[EXDEV]	Cross-device link.
8157	APPLICATION USAGE	
8158	Additional error numbers may be defined on conforming systems; see the System Interfaces	
8159	volume of IEEE Std. 1003.1-200x.	
8160	RATIONALE	
8161	None.	
8162	FUTURE DIRECTIONS	
8163	None.	
8164	SEE ALSO	
8165	The System Interfaces volume of IEEE Std. 1003.1-200x, Section 2.3, Error Numbers	
8166	CHANGE HISTORY	
8167	First released in Issue 1. Derived from Issue 1 of the SVID.	
8168	Issue 4	
8169	The [EILSEQ] error is added and marked as an EX interface.	
8170	The [ENOTBLK] error is withdrawn.	
8171	Issue 4, Version 2	
8172	The [EADDRINUSE], [EADDRNOTAVAIL], [EAFNOSUPPORT], [EALREADY], [EBADMSG],	
8173	[ECONNABORTED], [ECONNREFUSED], [ECONNRESET], [EDESTADDRREQ], [EDQUOT],	
8174	[EHOSTUNREACH], [EINPROGRESS], [EISCONN], [ELOOP], [EMSGSIZE], [EMULTIHOP],	
8175	[ENETDOWN], [ENETUNREACH], [ENOBUFS], [ENODATA], [ENOLINK],	

8176 [ENOPROTOOPT], [ENOSR], [ENOSTR], [ENOTCONN], [ENOTSOCK], [EOPNOTSUPP],
8177 [EOVERFLOW], [EPROTO], [EPROTONOSUPPORT], [EPROTOTYPE], [ESTALE], [ETIME],
8178 [ETIMEDOUT], and [EWOULDBLOCK] errors are added in the UX context.

8179 **Issue 5**

8180 Updated for alignment with the POSIX Realtime Extension.

8181 **Issue 6**

8182 The following new requirements on POSIX implementations derive from alignment with the
8183 Single UNIX Specification:

- 8184 • The majority of the error conditions previously marked as extensions are now mandatory,
8185 except for the STREAMS-related error conditions.

8186 **NAME**

8187 fcntl.h — file control options

8188 **SYNOPSIS**

8189 #include <fcntl.h>

8190 **DESCRIPTION**8191 The <fcntl.h> header shall define the following requests and arguments for use by the functions *fcntl()* and *open()*.8193 Values for *cmd* used by *fcntl()* (the following values are unique) are as follows:

8194 F_DUPFD Duplicate file descriptor.

8195 F_GETFD Get file descriptor flags.

8196 F_SETFD Set file descriptor flags.

8197 F_GETFL Get file status flags and file access modes.

8198 F_SETFL Set file status flags.

8199 F_GETLK Get record locking information.

8200 F_SETLK Set record locking information.

8201 F_SETLKW Set record locking information; wait if blocked.

8202 F_GETOWN Get process or process group ID to receive SIGURG signals.

8203 F_SETOWN Set process or process group ID to receive SIGURG signals.

8204 File descriptor flags used for *fcntl()* are as follows:8205 FD_CLOEXEC Close the file descriptor upon execution of an *exec* family function.8206 Values for *l_type* used for record locking with *fcntl()* (the following values are unique) are as follows:

8208 F_RDLCK Shared or read lock.

8209 F_UNLCK Unlock.

8210 F_WRLCK Exclusive or write lock.

8211 XSI The values used for *l_whence*, {SEEK_SET}, {SEEK_CUR}, and {SEEK_END} shall be defined as described in <unistd.h>.8213 The following four sets of values for *oflag* used by *open()* shall be bitwise-distinct:

8214 O_CREAT Create file if it does not exist.

8215 O_EXCL Exclusive use flag.

8216 O_NOCTTY Do not assign controlling terminal.

8217 O_TRUNC Truncate flag.

8218 File status flags used for *open()* and *fcntl()* are as follows:

8219 O_APPEND Set append mode.

8220 SIO O_DSYNC Write according to synchronized I/O data integrity completion.

8221 O_NONBLOCK Non-blocking mode.

8222 SIO O_RSYNC Synchronized read I/O operations.

8223 O_SYNC Write according to synchronized I/O file integrity completion.

8224 Mask for use with file access modes is as follows:

8225 O_ACCMODE Mask for file access modes.

8226 File access modes used for *open()* and *fcntl()* are as follows:

8227 O_RDONLY Open for reading only.

8228 O_RDWR Open for reading and writing.

8229 O_WRONLY Open for writing only.

8230 XSI The symbolic names for file modes for use as values of **mode_t** shall be defined as described in
8231 <sys/stat.h>.

8232 ADV Values for *advice* used by *posix_fadvise()* are as follows:

8233 POSIX_FADV_NORMAL
8234 The application has no advice to give on its behavior with respect to the specified data. It is
8235 the default characteristic if no advice is given for an open file.

8236 POSIX_FADV_SEQUENTIAL
8237 The application expects to access the specified data sequentially from lower offsets to
8238 higher offsets.

8239 POSIX_FADV_RANDOM
8240 The application expects to access the specified data in a random order.

8241 POSIX_FADV_WILLNEED
8242 The application expects to access the specified data in the near future.

8243 POSIX_FADV_DONTNEED
8244 The application expects that it will not access the specified data in the near future.

8245 POSIX_FADV_NOREUSE
8246 The application expects to access the specified data once and then not reuse it thereafter.
8247

8248 The structure **flock** describes a file lock. It shall include the following members:

8249 short l_type Type of lock; F_RDLCK, F_WRLCK, F_UNLCK.

8250 short l_whence Flag for starting offset.

8251 off_t l_start Relative offset in bytes.

8252 off_t l_len Size; if 0 then until EOF.

8253 pid_t l_pid Process ID of the process holding the lock; returned with F_GETLK.

8254 The **mode_t**, **off_t**, and **pid_t** types shall be defined as described in <sys/types.h>.

8255 The following shall be declared as functions and may also be defined as macros. Function
8256 prototypes shall be provided for use with an ISO C standard compiler.

8257 int creat(const char *, mode_t);

8258 int fcntl(int, int, ...);

8259 int open(const char *, int, ...);

8260 ADV int posix_fadvise(int, off_t, size_t, int);

8261 int posix_fallocate(int, off_t, size_t);

8262

- 8263 XSI Inclusion of the <fcntl.h> header may also make visible all symbols from <sys/stat.h> and
8264 <unistd.h>.
- 8265 **APPLICATION USAGE**
8266 None.
- 8267 **RATIONALE**
8268 None.
- 8269 **FUTURE DIRECTIONS**
8270 None.
- 8271 **SEE ALSO**
8272 <sys/stat.h>, <sys/types.h>, <unistd.h>, the System Interfaces volume of IEEE Std. 1003.1-200x,
8273 *creat()*, *exec()*, *fcntl()*, *open()*, *posix_fadvise()*, *posix_fallocate()*, *posix_madvise()*
- 8274 **CHANGE HISTORY**
8275 First released in Issue 1. Derived from Issue 1 of the SVID.
- 8276 **Issue 4**
8277 A reference to <unistd.h> is added for the definition of *l_whence*, {SEEK_SET}, {SEEK_CUR}, and
8278 {SEEK_END}, and marked as an extension.
8279 A reference to <sys/stat.h> is added for the symbolic names of file modes used as values of
8280 **mode_t**, and marked as an extension.
8281 A reference to <sys/types.h> is added for the definition of **mode_t**, **off_t**, and **pid_t**, and marked
8282 as an extension.
8283 A warning is added indicating that inclusion of <fcntl.h> may also make visible all symbols
8284 from <sys/stat.h> and <unistd.h>. This is marked as an extension.
8285 The following change is incorporated for alignment with the ISO POSIX-1 standard:
8286
 - The function declarations in this header are expanded to full ISO C standard prototypes.
- 8287 **Issue 5**
8288 The DESCRIPTION is updated for alignment with POSIX Realtime Extension.
- 8289 **Issue 6**
8290 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:
8291
 - O_DSYNC and O_RSYNC are marked as part of the Synchronized Input and Output option.
8292 The following new requirements on POSIX implementations derive from alignment with the
8293 Single UNIX Specification:
8294
 - The definition of the **mode_t**, **off_t**, and **pid_t** types is mandated.
8295 The F_GETOWN and F_SETOWN values are added for sockets.
8296 The *posix_fadvise()*, *posix_fallocate()*, and *posix_madvise()* functions are added for alignment with
8297 IEEE Std. 1003.1d-1999.
8298 IEEE PASC Interpretation 1003.1 #102 is applied moving the prototype for *posix_madvise()* to
8299 <sys_mman.h>.

8300 **NAME**8301 `fenv.h` — floating-point environment8302 **SYNOPSIS**8303 `#include <fenv.h>`8304 **DESCRIPTION**

8305 `CX` The functionality described on this reference page extends the ISO C standard. Applications
 8306 shall define the appropriate feature test macro (see the System Interfaces volume of
 8307 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 8308 symbols in this header.

8309 The **<fenv.h>** header shall define the following data types through **typedef**:

8310 **fenv_t** Represents the entire floating-point environment. The floating-point environment
 8311 refers collectively to any floating-point status flags and control modes supported
 8312 by the implementation.

8313 **fexcept_t** Represents the floating-point status flags collectively, including any status the
 8314 implementation associates with the flags. A floating-point status flag is a system
 8315 variable whose value is set (but never cleared) when a floating-point exception is
 8316 raised, which occurs as a side effect of exceptional floating-point arithmetic to
 8317 provide auxiliary information. A floating-point control mode is a system variable
 8318 whose value may be set by the user to affect the subsequent behavior of floating-
 8319 point arithmetic.

8320 The **<fenv.h>** header shall define the following constants:8321 `FE_DIVBYZERO`8322 `FE_INEXACT`8323 `FE_INVALID`8324 `FE_OVERFLOW`8325 `FE_UNDERFLOW`

8326 These constants are defined if and only if the implementation supports
 8327 the floating-point exception by means of the floating-point functions
 8328 `fwclearexcept()`, `fegetexceptflag()`, `feraiseexcept()`, `fesetexceptflag()`, and
 8329 `fetestexcept()`. Each expands to an integer constant expression with values
 8330 such that bitwise-inclusive ORs of all combinations of the constants result
 in distinct values.

8331 `FE_ALL_EXCEPT` Simply the bitwise-inclusive OR of all floating-point exception constants
 8332 defined above.

8333 `FE_DOWNWARD`8334 `FE_TONEAREST`8335 `FE_TOWARDZERO`8336 `FE_UPWARD`

8337 These constants are defined if and only if the implementation supports
 8338 getting and setting the represented rounding direction by means of the
 8339 `fegetround()` and `fesetround()` functions. Each expands to an integer
 constant expression whose values are distinct non-negative values.

8340 `FE_DFL_ENV`

8341 Represents the floating-point environment (that is, the one installed at
 8342 program startup) and has type pointer to const-qualified **fenv_t**. It can
 8343 be used as an argument to the functions within the **<fenv.h>** header that
 manage the floating-point environment.

8344 The following shall be declared as functions and may also be defined as macros. Function
 8345 prototypes shall be provided for use with an ISO C standard compiler.

```

8346 void feclearexcept(int);
8347 void fegetexceptflag(fexcept_t *, int);
8348 void feraiseexcept(int);
8349 void fesetexceptflag(const fexcept_t *, int);
8350 int fetestexcept(int);
8351 int fegetround(void);
8352 int fesetround(int);
8353 void fegetenv(fenv_t *);
8354 int feholdexcept(fenv_t *);
8355 void fesetenv(const fenv_t *);
8356 void feupdateenv(const fenv_t *);

```

8357 APPLICATION USAGE

8358 This header is designed to support the floating-point exception status flags and directed-
8359 rounding control modes required by the IEC 60559:1989 standard, and other similar floating-
8360 point state information. Also it is designed to facilitate code portability among all systems.

8361 Certain application programming conventions support the intended model of use for the
8362 floating-point environment:

- 8363 • A function call does not alter its caller's floating-point control modes, clear its caller's
8364 floating-point status flags, nor depend on the state of its caller's floating-point status flags
8365 unless the function is so documented.
- 8366 • A function call is assumed to require default floating-point control modes, unless its
8367 documentation promises otherwise.
- 8368 • A function call is assumed to have the potential for raising floating-point exceptions, unless
8369 its documentation promises otherwise.

8370 With these conventions, an application can safely assume default floating-point control modes
8371 (or be unaware of them). The responsibilities associated with accessing the floating-point
8372 environment fall on the application that does so explicitly.

8373 Even though the rounding direction macros may expand to constants corresponding to the
8374 values of FLT_ROUNDS, they are not required to do so.

8375 The FENV_ACCESS pragma provides a means to inform the implementation when an
8376 application might access the floating-point environment to test floating-point status flags or run
8377 under non-default floating-point control modes. The pragma shall occur either outside external
8378 declarations or preceding all explicit declarations and statements inside a compound statement.
8379 When outside external declarations, the pragma takes effect from its occurrence until another
8380 FENV_ACCESS pragma is encountered, or until the end of the translation unit. When inside a
8381 compound statement, the pragma takes effect from its occurrence until another FENV_ACCESS
8382 pragma is encountered (including within a nested compound statement), or until the end of the
8383 compound statement; at the end of a compound statement the state for the pragma is restored to
8384 its condition just before the compound statement. If this pragma is used in any other context, the
8385 behavior is undefined. If part of an application tests floating-point status flags, sets floating-
8386 point control modes, or runs under non-default mode settings, but was translated with the state
8387 for the FENV_ACCESS pragma off, the behavior is undefined. The default state (on or off) for
8388 the pragma is implementation-defined. (When execution passes from a part of the application
8389 translated with FENV_ACCESS off to a part translated with FENV_ACCESS on, the state of the
8390 floating-point status flags is unspecified and the floating-point control modes have their default
8391 settings.) For example:

```

8392 #include <fenv.h>
8393 void f(double x)

```

```
8394     {
8395         #pragma STDC FENV_ACCESS ON
8396         void g(double);
8397         void h(double);
8398         /* ... */
8399         g(x + 1);
8400         h(x + 1);
8401         /* ... */
8402     }
```

8403 If the function *g()* might depend on status flags set as a side effect of the first *x+1*, or if the
8404 second *x+1* might depend on control modes set as a side effect of the call to function *g()*, then
8405 the application shall contain an appropriately placed invocation as follows:

```
8406     #pragma STDC FENV_ACCESS ON
```

8407 **RATIONALE**

8408 The floating-point environment as defined here includes only execution-time modes, not the
8409 myriad of possible translation-time options that can affect an application's results. Each such
8410 option's deviation from IEEE Std. 1003.1-200x should be well documented.

8411 **Dynamic Versus Static Modes**

8412 Dynamic modes are potentially problematic because:

- 8413 1. The application may have to defend against undesirable mode settings, which impose
8414 intellectual as well as time and space overhead.
- 8415 2. The translator may not know which mode settings will be in effect or which functions
8416 change them at execution time, which inhibits optimization.

8417 The ISO/IEC 9899:1999 standard addresses these problems without changing the dynamic
8418 nature of the modes.

8419 An alternate approach would have been to present a model of static modes with explicit
8420 utterances to the translator about what mode settings would be in effect. This would have
8421 avoided any uncertainty due to the global nature of dynamic modes or the dependency on
8422 unenforced conventions. However, some essentially dynamic mechanism still would have been
8423 needed in order to allow functions to inherit (honor) their caller's modes. The IEC 60559:1989
8424 standard requires dynamic rounding direction modes. For the many architectures that maintain
8425 these modes in control registers, implementation of the static model would be more costly. Also,
8426 standard C has no facility, other than pragmas, for supporting static modes.

8427 An implementation on an architecture that provides only static control of modes (for example,
8428 through opword encodings) still could support the dynamic model, by generating multiple code
8429 streams with tests of a private global variable containing the mode setting. Only modules under
8430 an enabling `FENV_ACCESS` pragma would need such special treatment.

8431 **Translation**

8432 An implementation is not required to provide a facility for altering the modes for translation-
8433 time arithmetic, or for making exception flags from the translation available to the executing
8434 application. The language and library provide facilities to cause floating-point operations to be
8435 done at execution time when they can be subjected to varying dynamic modes and their
8436 exceptions detected. The need does not seem sufficient to require similar facilities for translation.

8437 **The fexcept_t Type**

8438 **fexcept_t** does not have to be an integer type. Its values must be obtained by a call to
 8439 *fegetexceptflag()*, and cannot be created by logical operations from the exception macros. An
 8440 implementation might simply implement **fexcept_** as an **int** and use the representations
 8441 reflected by the exception macros, but is not required to; other representations might contain
 8442 extra information about the exceptions. **fexcept_t** might be a **struct** with a member for each
 8443 exception (that might hold the address of the first or last floating-point instruction that caused
 8444 that exception). The ISO/IEC 9899:1999 standard makes no claims about the internals of an
 8445 **fexcept_t**, and so the user cannot inspect it.

8446 **Exception and Rounding Macros**

8447 Unsupported macros are not defined in order to ensure that their use results in a translation
 8448 error. An application might explicitly define such macros to allow translation of code (perhaps
 8449 never executed) containing the macros. An unsupported exception macro should be defined to
 8450 be 0; for example:

```
8451 #ifndef FE_INEXACT
8452 #define FE_INEXACT 0
8453 #endif
```

8454 so that a bitwise-inclusive OR of macros has a reasonable effect.

8455 **Exceptions**

8456 In previous drafts of IEEE Std. 1003.1-200x, several of the exception functions returned an **int**
 8457 indicating whether the *excepts* argument represented supported exceptions. This facility was
 8458 deemed unnecessary because:

```
8459 excepts & ~FE_ALL_EXCEPT
```

8460 can be used to test invalidity of the *excepts* argument.

8461 **Rounding Precision**

8462 The IEC 60559:1989 standard floating-point standard prescribes rounding precision modes (in
 8463 addition to the rounding direction modes covered by the functions in this reference page) as a
 8464 means for systems whose results are always double or extended to mimic systems that deliver
 8465 results to narrower formats. An implementation of C can meet this goal in any of the following
 8466 ways:

- 8467 1. By supporting the evaluation method indicated by `FLT_EVAL_METHOD` equal to 0
- 8468 2. By providing pragmas or compile options to shorten results by rounding to the
 8469 IEC 60559:1989 standard single or double precision
- 8470 3. By providing functions to dynamically set and get rounding precision modes which
 8471 shorten results by rounding to the IEC 60559:1989 standard single or double precision;
 8472 recommended are functions *fesetprec()* and *fegetprec()* and macros `FE_FLTPREC`,
 8473 `FE_DBLPREC`, and `FE_LDBLPREC`, analogous to the functions and macros for the
 8474 rounding direction modes

8475 IEEE Std. 1003.1-200x does not include a portable interface for precision control because the
 8476 IEC 60559:1989 standard floating-point standard is ambivalent on whether it intends for
 8477 precision control to be dynamic (like the rounding direction modes) or static. Indeed, some
 8478 floating-point architectures provide control modes suitable for a dynamic mechanism, and
 8479 others rely on instructions to deliver single and double-format results suitable only for a static

8480 mechanism.

8481 **FUTURE DIRECTIONS**

8482 None.

8483 **SEE ALSO**

8484 The System Interfaces volume of IEEE Std. 1003.1-200x, *feclearexcept()*, *fegetenv()*,
8485 *fegetexceptflag()*, *fegetround()*, *fehldexcept()*, *feraiseexcept()*, *fesetenv()*, *fesetexceptflag()*,
8486 *fesetround()*, *fetestexcept()*, *feupdateenv()*

8487 **CHANGE HISTORY**

8488 First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.

8489 **NAME**

8490 float.h — floating types

8491 **SYNOPSIS**

8492 #include <float.h>

8493 **DESCRIPTION**

8494 **cx** 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 symbols in this header.

8498 The characteristics of floating types are defined in terms of a model that describes a representation of floating-point numbers and values that provide information about an implementation's floating-point arithmetic.

8501 The following parameters are used to define the model for each floating-point type:

8502 *s* Sign (± 1).

8503 *b* Base or radix of exponent representation (an integer > 1).

8504 *e* Exponent (an integer between a minimum e_{\min} and a maximum e_{\max}).

8505 *p* Precision (the number of base-*b* digits in the significand).

8506 f_k Non-negative integers less than *b* (the significand digits).

8507 A normalized floating-point number x ($f_1 > 0$ if $x \neq 0$) is defined by the following model:

8508
$$x = s \times b^e \times \sum_{k=1}^p f_k \times b^{-k}, e_{\min} \leq e \leq e_{\max}$$

8509

8510 FLT_RADIX is a constant expression suitable for use in the #if preprocessing directives. All constants except FLT_RADIX and FLT_ROUNDSS have separate names for all three floating-point types. The floating-point model representation is provided for all macro names except FLT_ROUNDSS.

8514 The rounding mode for floating-point addition is characterized by the value of FLT_ROUNDSS:

8515 -1 Indeterminable.

8516 0 Toward 0.0.

8517 1 To nearest.

8518 2 Toward positive infinity.

8519 3 Toward negative infinity.

8520 All other values for FLT_ROUNDSS characterize implementation-defined rounding behavior.

8521 The values of operations with floating operands and values subject to the usual arithmetic conversions and of floating constants are evaluated to a format whose range and precision may be greater than required by the type. The use of evaluation formats is characterized by the implementation-defined value of FLT_EVAL_METHOD:

8525 -1 Indeterminable.

8526 0 Evaluate all operations and constants just to the range and precision of the type.

8527 1 Evaluate operations and constants of type **float** and **double** to the range and precision of the **double** type, evaluate **long double** operations and constants to the range and precision of the **long double** type.

8530 2 Evaluate all operations and constants to the range and precision of the **long double** type. |
8531 All other negative values for FLT_EVAL_METHOD characterize implementation-defined |
8532 behavior. |
8533 The macro names given in the following list are defined as expressions with values that are |
8534 equal or greater in magnitude (absolute value) to those shown, with the same sign. |

Name	Description	Value
FLT_RADIX	Radix of exponent representation, b .	2
FLT_MANT_DIG DBL_MANT_DIG LDBL_MANT_DIG	Number of base-FLT_RADIX digits in the floating-point significand, p . † †	†
DECIMAL_DIG	Number of decimal digits, n , such that any floating-point number in the widest supported floating type with p_{max} radix b digits can be rounded to a floating-point number with n decimal digits and back again without change to the value. Notes to Reviewers <i>This section with side shading will not appear in the final copy. - Ed.</i> D3, XSH, ERN 146 requires a new equation to be inserted here. However, none of the equations in float.h match the C99 style. This needs looking at again.	10
FLT_DIG DBL_DIG LDBL_DIG	Number of decimal digits, q , such that any floating-point number with q decimal digits can be rounded into a floating-point number with p radix b digits and back again without change to the q decimal digits, $\left\lfloor (p-1) \times \log_{10} b \right\rfloor + \begin{cases} 1 & \text{if } b \text{ is a power of } 10 \\ 0 & \text{otherwise} \end{cases}$	6 10 10
FLT_MIN_EXP DBL_MIN_EXP LDBL_MIN_EXP	Minimum negative integer such that FLT_RADIX raised to that power minus 1 is a normalized floating-point number, e_{min} † †	†
FLT_MIN_10_EXP DBL_MIN_10_EXP LDBL_MIN_10_EXP	Minimum negative integer such that 10 raised to that power is in the range of normalized floating-point numbers, $\left\lfloor \log_{10} b^{e_{min}^{-1}} \right\rfloor$	-37 -37 -37
FLT_MAX_EXP DBL_MAX_EXP LDBL_MAX_EXP	Maximum integer such that FLT_RADIX raised to that power minus 1 is a representable finite floating-point number, e_{max} † †	†

8535	FLT_MAX_10_EXP	Maximum integer such that 10 raised to that power is in the range of representable finite floating-point numbers,	37
		$\left\lfloor \log_{10}((1 - b^{-p}) \times b^{e_{\max}}) \right\rfloor$	
	DBL_MAX_10_EXP		37
	LDBL_MAX_10_EXP		37

8536 † Implementation-defined values.

8537 The macro names given in the following list are defined as expressions with values that are
8538 equal to or greater than those shown.

8539	FLT_MAX	Maximum representable finite floating-point number,	1E+37
		$(1 - b^{-p}) \times b^{e_{\max}}$	
	DBL_MAX		1E+37
	LDBL_MAX		1E+37

8540 The macro names given in the following list are defined as expressions with values that are
8541 equal to or less than those shown.

8542	FLT_EPSILON	The difference between 1.0 and the least value greater than 1.0 that is representable in the given floating-point type,	1E-5
		$b^{(1-p)}$	
	DBL_EPSILON		1E-9
	LDBL_EPSILON		1E-9
	FLT_MIN	Minimum normalized positive floating-point number,	1E-37
		$b^{(e_{\min} - 1)}$	
	DBL_MIN		1E-37
	LDBL_MIN		1E-37

8543 **APPLICATION USAGE**

8544 None.

8545 **RATIONALE**

8546 None.

8547 **FUTURE DIRECTIONS**

8548 None.

8549 **SEE ALSO**

8550 None.

8551 **CHANGE HISTORY**

8552 First released in Issue 4. Derived from the ISO C standard.

8553 **Issue 6**

8554 The description of the operations with floating-point values is updated for alignment with the
8555 ISO/IEC 9899:1999 standard.

8556 **NAME**

8557 fmtmsg.h — message display structures

8558 **SYNOPSIS**

8559 XSI #include <fmtmsg.h>

8560

8561 **DESCRIPTION**

8562 The <fmtmsg.h> header shall define the following macros, which expand to constant integral
8563 expressions:

- 8564 MM_HARD Source of the condition is hardware.
- 8565 MM_SOFT Source of the condition is software.
- 8566 MM_FIRM Source of the condition is firmware.
- 8567 MM_APPL Condition detected by application.
- 8568 MM_UTIL Condition detected by utility.
- 8569 MM_OPSYS Condition detected by operating system.
- 8570 MM_RECOVER Recoverable error.
- 8571 MM_NRECOV Non-recoverable error.
- 8572 MM_HALT Error causing application to halt.
- 8573 MM_ERROR Application has encountered a non-fatal fault.
- 8574 MM_WARNING Application has detected unusual non-error condition.
- 8575 MM_INFO Informative message.
- 8576 MM_NOSEV No severity level provided for the message.
- 8577 MM_PRINT Display message on standard error.
- 8578 MM_CONSOLE Display message on system console.

8579 The table below indicates the null values and identifiers for *fmtmsg()* arguments. The
8580 <fmtmsg.h> header shall define the macros in the **Identifier** column, which expand to constant
8581 expressions that expand to expressions of the type indicated in the **Type** column:

8582

8583

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

8584

8585

8586

8587

8588

8589

8590 The <fmtmsg.h> header shall also define the following macros for use as return values for
8591 *fmtmsg()*:

- 8592 MM_OK The function succeeded.
- 8593 MM_NOTOK The function failed completely.
- 8594 MM_NOMSG The function was unable to generate a message on standard error, but
8595 otherwise succeeded.

8596 MM_NOCON The function was unable to generate a console message, but otherwise
8597 succeeded.

8598 The following shall be declared as a function and may also be defined as a macro. A function
8599 prototype shall be provided for use with an ISO C standard compiler.

```
8600           int fmtmsg(long, const char *, int,  
8601                const char *, const char *, const char *);
```

8602 **APPLICATION USAGE**

8603 None.

8604 **RATIONALE**

8605 None.

8606 **FUTURE DIRECTIONS**

8607 None.

8608 **SEE ALSO**

8609 The System Interfaces volume of IEEE Std. 1003.1-200x, *fmtmsg()*

8610 **CHANGE HISTORY**

8611 First released in Issue 4, Version 2.

8612 **NAME**

8613 fnmatch.h — file name-matching types

8614 **SYNOPSIS**

8615 #include <fnmatch.h>

8616 **DESCRIPTION**8617 The <fnmatch.h> header shall define the flags and return value used by the *fnmatch()* function.

8618 The following constants are defined:

8619 FNM_NOMATCH The string does not match the specified pattern.

8620 FNM_PATHNAME Slash in *string* only matches slash in *pattern*.8621 FNM_PERIOD Leading period in *string* must be exactly matched by period in *pattern*.

8622 FNM_NOESCAPE Disable backslash escaping.

8623 FNM_NOSYS The implementation does not support this function. **(LEGACY)**8624 The following shall be declared as a function and may also be declared as a macro. Function
8625 prototypes shall be provided for use with an ISO C standard compiler.

8626 int fnmatch(const char *, const char *, int);

8627 **APPLICATION USAGE**

8628 None.

8629 **RATIONALE**

8630 None.

8631 **FUTURE DIRECTIONS**

8632 None.

8633 **SEE ALSO**8634 The System Interfaces volume of IEEE Std. 1003.1-200x, *fnmatch()*, the Shell and Utilities volume
8635 of IEEE Std. 1003.1-200x8636 **CHANGE HISTORY**

8637 First released in Issue 4. Derived from the ISO POSIX-2 standard.

8638 **Issue 6**

8639 The constant FNM_NOSYS is marked LEGACY.

8640 **NAME**8641 `ftw.h` — file tree traversal8642 **SYNOPSIS**8643 XSI `#include <ftw.h>`

8644

8645 **DESCRIPTION**8646 The `<ftw.h>` header shall define the **FTW** structure that includes at least the following members:8647 `int base`8648 `int level`8649 The `<ftw.h>` header shall define macros for use as values of the third argument to the
8650 application-supplied function that is passed as the second argument to `ftw()` and `nftw()`:8651 `FTW_F` File.8652 `FTW_D` Directory.8653 `FTW_DNR` Directory without read permission.8654 `FTW_DP` Directory with subdirectories visited.8655 `FTW_NS` Unknown type; `stat()` failed.8656 `FTW_SL` Symbolic link.8657 `FTW_SLN` Symbolic link that names a nonexistent file.8658 The `<ftw.h>` header shall define macros for use as values of the fourth argument to `nftw()`:8659 `FTW_PHYS` Physical walk, does not follow symbolic links. Otherwise, `nftw()` follows
8660 links but does not walk down any path that crosses itself.8661 `FTW_MOUNT` The walk does not cross a mount point.8662 `FTW_DEPTH` All subdirectories are visited before the directory itself.8663 `FTW_CHDIR` The walk changes to each directory before reading it.8664 The following shall be declared as functions and may also be defined as macros. Function
8665 prototypes shall be provided for use with an ISO C standard compiler.8666 `int ftw(const char *,`
8667 `int (*)(const char *, const struct stat *, int), int);`
8668 `int nftw(const char *, int (*)`
8669 `(const char *, const struct stat *, int, struct FTW*),`
8670 `int, int);`8671 The `<ftw.h>` header shall define the **stat** structure and the symbolic names for `st_mode` and the
8672 file type test macros as described in `<sys/stat.h>`.8673 Inclusion of the `<ftw.h>` header may also make visible all symbols from `<sys/stat.h>`.

8674 **APPLICATION USAGE**

8675 None.

8676 **RATIONALE**

8677 None.

8678 **FUTURE DIRECTIONS**

8679 None.

8680 **SEE ALSO**8681 <sys/stat.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *ftw()*, *nftw()*8682 **CHANGE HISTORY**

8683 First released in Issue 1. Derived from Issue 1 of the SVID.

8684 **Issue 4**

8685 The function declarations in this header are expanded to full ISO C standard prototypes.

8686 A reference to <sys/stat.h> is added for the definition of the **stat** structure, the symbolic names
8687 for *st_mode*, and the file type test macros.8688 A warning is added indicating that inclusion of <ftw.h> may also make visible all symbols from
8689 <sys/stat.h>.8690 **Issue 4, Version 2**

8691 The following changes are incorporated in the DESCRIPTION for X/OPEN UNIX conformance:

- 8692 • The **FTW** structure is defined.
- 8693 • The *nftw()* function is declared by the header and is mentioned as one of the functions to
8694 which the first list of macros applies.
- 8695 • FTW_SL and FTW_SLN are added to the first list of macros to handle symbolic links.
- 8696 • Macros for use as values of the fourth argument to *nftw()* are defined.

8697 **Issue 5**

8698 A description of FTW_DP is added.

8699 **NAME**8700 `glob.h` — path name pattern-matching types8701 **SYNOPSIS**8702 `#include <glob.h>`8703 **DESCRIPTION**8704 The `<glob.h>` header shall define the structures and symbolic constants used by the `glob()`
8705 function.8706 The structure type `glob_t` shall contain at least the following members:8707 `size_t gl_pathc` Count of paths matched by *pattern*.8708 `char **gl_pathv` Pointer to a list of matched path names.8709 `size_t gl_offs` Slots to reserve at the beginning of *gl_pathv*.8710 The following constants shall be provided as values for the *flags* argument:8711 `GLOB_APPEND` Append generated path names to those previously obtained.8712 `GLOB_DOOFFS` Specify how many null pointers to add to the beginning of *pglob-*
8713 *>gl_pathv*.8714 `GLOB_ERR` Cause `glob()` to return on error.8715 `GLOB_MARK` Each path name that is a directory that matches *pattern* has a slash
8716 appended.8717 `GLOB_NOCHECK` If *pattern* does not match any path name, then return a list consisting of
8718 only *pattern*.8719 `GLOB_NOESCAPE` Disable backslash escaping.8720 `GLOB_NOSORT` Do not sort the path names returned.

8721 The following constants shall be defined as error return values:

8722 `GLOB_ABORTED` The scan was stopped because `GLOB_ERR` was set or `(*errfunc)()`
8723 returned non-zero.8724 `GLOB_NOMATCH` The *pattern* does not match any existing path name, and
8725 `GLOB_NOCHECK` was not set in flags.8726 `GLOB_NOSPACE` An attempt to allocate memory failed.8727 `GLOB_NOSYS` The implementation does not support this function.8728 The following shall be declared as functions and may also be declared as macros. Function
8729 prototypes shall be provided for use with an ISO C standard compiler.8730 `int glob(const char *restrict, int, int (*restrict)(const char *, int),`
8731 `glob_t *restrict);`8732 `void globfree (glob_t *);`8733 The implementation may define additional macros or constants using names beginning with
8734 `GLOB_`.

8735 **APPLICATION USAGE**

8736 None.

8737 **RATIONALE**

8738 None.

8739 **FUTURE DIRECTIONS**

8740 None.

8741 **SEE ALSO**8742 The System Interfaces volume of IEEE Std. 1003.1-200x, *glob()*, the Shell and Utilities volume of
8743 IEEE Std. 1003.1-200x8744 **CHANGE HISTORY**

8745 First released in Issue 4. Derived from the ISO POSIX-2 standard.

8746 **Issue 6**8747 The **restrict** keyword is added to the prototype for *glob()*.

8748 **NAME**

8749 grp.h — group structure

8750 **SYNOPSIS**

8751 #include <grp.h>

8752 **DESCRIPTION**8753 The **<grp.h>** header shall declare the structure **group** which shall include the following
8754 members:

8755 char *gr_name The name of the group.
 8756 gid_t gr_gid Numerical group ID.
 8757 char **gr_mem Pointer to a null-terminated array of character
 8758 pointers to member names.

8759 The **gid_t** type shall be defined as described in **<sys/types.h>**.8760 The following shall be declared as functions and may also be defined as macros. Function
8761 prototypes shall be provided for use with an ISO C standard compiler.

```
8762       struct group *getgrgid(gid_t);
8763       struct group *getgrnam(const char *);
8764 TSF       int        getgrgid_r(gid_t, struct group *, char *,
8765                               size_t, struct group **);
8766       int        getgrnam_r(const char *, struct group *, char *,
8767                               size_t , struct group **);
8768 XSI       struct group *getgrent(void);
8769       void       endgrent(void);
8770       void       setgrent(void);
```

8771

8772 **APPLICATION USAGE**

8773 None.

8774 **RATIONALE**

8775 None.

8776 **FUTURE DIRECTIONS**

8777 None.

8778 **SEE ALSO**8779 **<sys/types.h>**, the System Interfaces volume of IEEE Std. 1003.1-200x, *endgrent()*, *getgrgid()*,
8780 *getgrnam()*8781 **CHANGE HISTORY**

8782 First released in Issue 1.

8783 **Issue 4**8784 A reference to **<sys/types.h>** is added for the definition of **gid_t** and marked as an extension.

8785 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 8786
- The function declarations in this header are expanded to full ISO C standard prototypes.

8787 **Issue 4, Version 2**8788 For X/OPEN UNIX conformance, the *getgrent()*, *endgrent()*, and *setgrent()* functions are added
8789 to the list of functions declared in this header.

8790 **Issue 5**

8791 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

8792 **Issue 6**

8793 The following new requirements on POSIX implementations derive from alignment with the
8794 Single UNIX Specification:

- 8795 • The definition of **gid_t** is mandated.
- 8796 • The *getgrgid_r()* and *getgrnam_r()* functions are marked as part of the Thread-Safe Functions
8797 option.

8798 **NAME**

8799 iconv.h — codeset conversion facility

8800 **SYNOPSIS**8801 XSI `#include <iconv.h>`

8802

8803 **DESCRIPTION**8804 The **<iconv.h>** header shall define the following data type through **typedef**:8805 **iconv_t** Identifies the conversion from one codeset to another.8806 The following shall be declared as functions and may also be declared as macros. Function
8807 prototypes shall be provided for use with an ISO C standard compiler.8808 `iconv_t iconv_open(const char *, const char *);`8809 `size_t iconv(iconv_t, char **restrict, size_t *restrict, char **restrict,`
8810 `size_t *restrict);`8811 `int iconv_close(iconv_t);`8812 **APPLICATION USAGE**

8813 None.

8814 **RATIONALE**

8815 None.

8816 **FUTURE DIRECTIONS**

8817 None.

8818 **SEE ALSO**8819 The System Interfaces volume of IEEE Std. 1003.1-200x, *iconv()*, *iconv_close()*, *iconv_open()*8820 **CHANGE HISTORY**

8821 First released in Issue 4.

8822 **Issue 6**8823 The **restrict** keyword is added to the prototype for *iconv()*.

8824 NAME

8825 inttypes.h — fixed size integer types

8826 SYNOPSIS

8827 XSI `#include <inttypes.h>`
8828

8829 DESCRIPTION

8830 The <inttypes.h> header shall include the <stdint.h> header.

8831 **Notes to Reviewers**8832 *This section with side shading will not appear in the final copy. - Ed.*8833 Reviewers are asked to propose changes to eliminate duplication between inttypes.h and
8834 stdint.h.

8835 The <inttypes.h> header shall include definitions of at least the following types:

8836 **imaxdiv_t** Structure type that is the type of the value returned by the *imaxdiv()* function.8837 **int8_t** 8-bit signed integer type.8838 **int16_t** 16-bit signed integer type.8839 **int32_t** 32-bit signed integer type.8840 **uint8_t** 8-bit unsigned integer type.8841 **uint16_t** 16-bit unsigned integer type.8842 **uint32_t** 32-bit unsigned integer type.8843 **intptr_t** Signed integer type large enough to hold any pointer.8844 **uintptr_t** Unsigned integer type large enough to hold any pointer.

8845 If any of the following are true:

8846 • The implementation supports the `_POSIX_V6_ILP32_OFFBIG` programming environment
8847 and the application is being built in the `_POSIX_V6_ILP32_OFFBIG` programming
8848 environment (see the Shell and Utilities volume of IEEE Std. 1003.1-200x, *c99*, Programming
8849 Environments).8850 • The implementation supports the `_POSIX_V6_LP64_OFF64` programming environment and
8851 the application is being built in the `_POSIX_V6_LP64_OFF64` programming environment.8852 • The implementation supports the `_POSIX_V6_LPBIG_OFFBIG` programming environment
8853 and the application is being built in the `_POSIX_V6_LPBIG_OFFBIG` programming
8854 environment.

8855 then <inttypes.h> also shall include definitions for the following types:

8856 **int64_t** 64-bit signed integer type.8857 **uint64_t** 64-bit unsigned integer type.8858 If `__STDC_FORMAT_MACROS` is defined before <inttypes.h> is included, then the following
8859 object-like macros shall be defined. Each expands to a character string literal containing a
8860 conversion specifier, possibly modified by a length modifier, suitable for use within the *format*
8861 argument of a formatted input/output function when converting the corresponding integer
8862 type. These macro names have the general form of PRI (character string literals for the *fprintf()*
8863 and *fwprintf()* family of functions) or SCN (character string literals for the *fscanf()* and *fwscanf()*)

8864 family of functions), followed by the conversion specifier, followed by a name corresponding to
 8865 a similar type name in <stdint.h>. In these names, N represents the width of the type as
 8866 described in *stdint.h*(. For example, *PRIdFAST32* can be used in a format string to print the
 8867 value of an integer of type *int_fast32_t*.

8868 The *fprintf*() macros for signed integers are:

8869	PRIdN	PRIdLEASTN	PRIdFASTN	PRIdMAX	PRIdPTR
8870	PRiIN	PRiILEASTN	PRiIFASTN	PRiIMAX	PRiIPTR

8871 The *fprintf*() macros for unsigned integers are:

8872	PRIoN	PRIoLEASTN	PRIoFASTN	PRIoMAX	PRIoPTR
8873	PRiUN	PRiULEASTN	PRiUFASTN	PRiUMAX	PRiUPTR
8874	PRiXN	PRiXLEASTN	PRiXFASTN	PRiXMAX	PRiXPTR
8875	PRiXN	PRiXLEASTN	PRiXFASTN	PRiXMAX	PRiXPTR

8876 The *fscanf*() macros for signed integers are:

8877	SCNdN	SCNdLEASTN	SCNdFASTN	SCNdMAX	SCNdPTR
8878	SCNiN	SCNiLEASTN	SCNiFASTN	SCNiMAX	SCNiPTR

8879 The *fscanf*() macros for unsigned integers are:

8880	SCNoN	SCNoLEASTN	SCNoFASTN	SCNoMAX	SCNoPTR
8881	SCNuN	SCNuLEASTN	SCNuFASTN	SCNuMAX	SCNuPTR
8882	SCNxN	SCNxLEASTN	SCNxFASTN	SCNxMAX	SCNxPTR

8883 For each type that the implementation provides in <stdint.h>, the corresponding *fprintf*()
 8884 macros shall be defined and the corresponding *fscanf*() macros shall be defined unless the
 8885 implementation does not have a suitable *fscanf* length modifier for the type.

8886 The following shall be declared as functions and may also be defined as macros. Function
 8887 prototypes shall be provided for use with an ISO C standard compiler.

```
8888 intmax_t imaxabs(intmax_t);
8889 imaxdiv_t imaxdiv(intmax_t, intmax_t);
8890 intmax_t strtoumax(const char *restrict, char **restrict, int);
8891 uintmax_t strtoumax(const char *restrict, char **restrict, int);
8892 intmax_t wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);
8893 uintmax_t wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);
```

8894 **EXAMPLES**

```
8895 #include <inttypes.h>
8896 #include <wchar.h>
8897 int main(void)
8898 {
8899     uintmax_t i = UINTMAX_MAX; // This type always exists.
8900     wprintf(L"The largest integer value is %020"
8901           PRIxMAX "\n", i);
8902     return 0;
8903 }
```

8904 **APPLICATION USAGE**

8905 None.

8906 **RATIONALE**

8907 The <inttypes.h> header was derived from the header of the same name found on several
8908 existing 64-bit systems. The C Standard Committee debated other methods for specifying
8909 integer sizes and other characteristics, but in the end decided to standardize existing practice
8910 rather than innovate in this area.

8911 The ISO/IEC 9899:1990 standard specifies that the language should support four signed and
8912 unsigned integer data types—**char**, **short**, **int**, and **long**—but places very little requirement on
8913 their size other than that **int** and **short** be at least 16 bits and **long** be at least as long as **int** and
8914 not smaller than 32 bits. For 16-bit systems, most implementations assign 8, 16, 16, and 32 bits to
8915 **char**, **short**, **int**, and **long**, respectively. For 32-bit systems, the common practice is to assign 8, 16,
8916 32, and 32 bits to these types. This difference in **int** size can create some problems for users who
8917 migrate from one system to another which assigns different sizes to integer types, because the
8918 ISO C standard integer promotion rule can produce silent changes unexpectedly. The need for
8919 defining an extended integer type increased with the introduction of 64-bit systems.

8920 The purpose of <inttypes.h> is to provide a set of integer types whose definitions are consistent
8921 across machines and independent of operating systems and other implementation
8922 idiosyncrasies. It defines, via **typedef**, integer types of various sizes. Implementations are free to
8923 **typedef** them as ISO C standard integer types or extensions that they support. Consistent use of
8924 this header will greatly increase the portability of a users program across platforms.

8925 **FUTURE DIRECTIONS**

8926 Macro names beginning with PRI or SCN followed by any lowercase letter or 'x' may be added
8927 to the macros defined in the <inttypes.h> header.

8928 **SEE ALSO**8929 The System Interfaces volume of IEEE Std. 1003.1-200x, *imaxdiv()*8930 **CHANGE HISTORY**

8931 First released in Issue 5.

8932 **Issue 6**

8933 The Open Group Base Resolution bwg97-006 is applied.

8934 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

8935 **NAME**

8936 iso646.h — alternative spellings

8937 **SYNOPSIS**

8938 #include <iso646.h>

8939 **DESCRIPTION**

8940 **cx** The functionality described on this reference page extends the ISO C standard. Applications
8941 shall define the appropriate feature test macro (see the System Interfaces volume of
8942 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
8943 symbols in this header.

8944 The **<iso646.h>** header shall define the following eleven macros (on the left) that expand to the
8945 corresponding tokens (on the right):

8946 *and* &&8947 *and_eq* &=8948 *bitand* &8949 *bitor* |8950 *compl* ~8951 *not* !8952 *not_eq* !=8953 *or* | |8954 *or_eq* |=8955 *xor* ^8956 *xor_eq* ^=8957 **APPLICATION USAGE**

8958 None.

8959 **RATIONALE**

8960 None.

8961 **FUTURE DIRECTIONS**

8962 None.

8963 **SEE ALSO**

8964 None.

8965 **CHANGE HISTORY**

8966 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

8967 **NAME**

8968 langinfo.h — language information constants

8969 **SYNOPSIS**

8970 xSI #include <langinfo.h>

8971

8972 **DESCRIPTION**

8973 The <langinfo.h> header contains the constants used to identify items of *langinfo* data (see
8974 *nl_langinfo()*). The type of the constant, **nl_item**, shall be defined as described in <nl_types.h>.

8975 The following constants shall be defined. The entries under **Category** indicate in which
8976 *setlocale()* category each item is defined.

8977

8978

Constant	Category	Meaning
CODESET	LC_CTYPE	Codeset name.
D_T_FMT	LC_TIME	String for formatting date and time.
D_FMT	LC_TIME	Date format string.
T_FMT	LC_TIME	Time format string.
T_FMT_AMPM	LC_TIME	a.m. or p.m. time format string.
AM_STR	LC_TIME	Ante Meridian affix.
PM_STR	LC_TIME	Post Meridian affix.
DAY_1	LC_TIME	Name of the first day of the week (for example, Sunday).
DAY_2	LC_TIME	Name of the second day of the week (for example, Monday).
DAY_3	LC_TIME	Name of the third day of the week (for example, Tuesday).
DAY_4	LC_TIME	Name of the fourth day of the week (for example, Wednesday).
DAY_5	LC_TIME	Name of the fifth day of the week (for example, Thursday).
DAY_6	LC_TIME	Name of the sixth day of the week (for example, Friday).
DAY_7	LC_TIME	Name of the seventh day of the week (for example, Saturday).
ABDAY_1	LC_TIME	Abbreviated name of the first day of the week.
ABDAY_2	LC_TIME	Abbreviated name of the second day of the week.
ABDAY_3	LC_TIME	Abbreviated name of the third day of the week.
ABDAY_4	LC_TIME	Abbreviated name of the fourth day of the week.
ABDAY_5	LC_TIME	Abbreviated name of the fifth day of the week.
ABDAY_6	LC_TIME	Abbreviated name of the sixth day of the week.
ABDAY_7	LC_TIME	Abbreviated name of the seventh day of the week.
MON_1	LC_TIME	Name of the first month of the year.
MON_2	LC_TIME	Name of the second month.
MON_3	LC_TIME	Name of the third month.
MON_4	LC_TIME	Name of the fourth month.
MON_5	LC_TIME	Name of the fifth month.
MON_6	LC_TIME	Name of the sixth month.
MON_7	LC_TIME	Name of the seventh month.
MON_8	LC_TIME	Name of the eighth month.
MON_9	LC_TIME	Name of the ninth month.
MON_10	LC_TIME	Name of the tenth month.
MON_11	LC_TIME	Name of the eleventh month.
MON_12	LC_TIME	Name of the twelfth month.

9013

9014
9015
9016
9017
9018
9019
9020
9021
9022
9023
9024
9025
9026
9027
9028
9029
9030
9031
9032
9033
9034
9035
9036
9037
9038
9039
9040

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.

9041 If the locale's value for **p_cs_precedes** and **n_cs_precedes** does not match, the value of
9042 *nl_langinfo*(CRNCYSTR) is unspecified.

9043 The following shall be declared as a function and may also be declared as a macro. Function
9044 prototypes shall be provided for use with an ISO C standard compiler.

9045

```
char *nl_langinfo(nl_item);
```

9046 Inclusion of the **<langinfo.h>** header may also make visible all symbols from **<nl_types.h>**.

9047 APPLICATION USAGE

9048 Wherever possible, users are advised to use functions compatible with those in the ISO C
9049 standard to access items of *langinfo* data. In particular, the *strptime*() function should be used to
9050 access date and time information defined in category *LC_TIME*. The *localeconv*() function
9051 should be used to access information corresponding to *RADIXCHAR*, *THOUSEP*, and
9052 *CRNCYSTR*.

9053 RATIONALE

9054 None.

9055 FUTURE DIRECTIONS

9056 None.

9057 SEE ALSO

9058 The System Interfaces volume of IEEE Std. 1003.1-200x, *nl_langinfo*(), *localeconv*(), *strfmon*(),
9059 *strptime*(), Chapter 7 (on page 143)

9060 **CHANGE HISTORY**

9061 First released in Issue 2.

9062 **Issue 4**

9063 The function declarations in this header are expanded to full ISO C standard prototypes.

9064 The constants CODESET, T_FMT_AMPM, ERA, ERA_D_FMT, ALT_DIGITS, YESEXPR, and
9065 NOEXPR are added.

9066 The constants YESSTR and NOSTR are marked TO BE WITHDRAWN.

9067 Reference to the Gregorian calendar is removed.

9068 The constants YESSTR and NOSTR are now defined as belonging to category *LC_MESSAGES*.
9069 Previously they were defined as constants in category *LC_ALL*.9070 A warning is added indicating that inclusion of <langinfo.h> may also make visible all symbols
9071 from <nl_types.h>.9072 The APPLICATION USAGE section is expanded to recommend use of the *localeconv()* function.9073 **Issue 5**

9074 The constants YESSTR and NOSTR are marked LEGACY.

9075 **Issue 6**

9076 The constants YESSTR and NOSTR are removed.

9077 **NAME**

9078 libgen.h — definitions for pattern matching functions

9079 **SYNOPSIS**

9080 xSI #include <libgen.h>

9081

9082 **DESCRIPTION**9083 The following shall be declared as functions and may also be defined as macros. Function
9084 prototypes shall be provided for use with an ISO C standard compiler.

9085 char *basename(char *);

9086 char *dirname(char *);

9087 **APPLICATION USAGE**

9088 None.

9089 **RATIONALE**

9090 None.

9091 **FUTURE DIRECTIONS**

9092 None.

9093 **SEE ALSO**9094 The System Interfaces volume of IEEE Std. 1003.1-200x, *basename()*, *dirname()*9095 **CHANGE HISTORY**

9096 First released in Issue 4, Version 2.

9097 **Issue 5**9098 The function prototypes for *basename()* and *dirname()* are changed to indicate that the first
9099 argument is of type **char*** rather than **const char***.

9100 **NAME**

9101 limits.h — implementation-defined constants

9102 **SYNOPSIS**

9103 #include <limits.h>

9104 **DESCRIPTION**

9105 cx The functionality described on this reference page extends the ISO C standard. Applications
 9106 shall define the appropriate feature test macro (see the System Interfaces volume of
 9107 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 9108 symbols in this header.

9109 The <limits.h> header shall define various symbolic names. Different categories of names are
 9110 described below.

9111 The names represent various limits on resources that the implementation imposes on
 9112 applications.

9113 Implementations may choose any appropriate value for each limit, provided it is not more
 9114 restrictive than the Minimum Acceptable Values listed below. Symbolic constant names
 9115 beginning with _POSIX may be found in <unistd.h>.

9116 Applications should not assume any particular value for a limit. To achieve maximum
 9117 portability, an application should not require more resource than the Minimum Acceptable
 9118 Value quantity. However, an application wishing to avail itself of the full amount of a resource
 9119 available on an implementation may make use of the value given in <limits.h> on that
 9120 particular implementation, by using the symbolic names listed below. It should be noted,
 9121 however, that many of the listed limits are not invariant, and at runtime, the value of the limit
 9122 may differ from those given in this header, for the following reasons:

- 9123 • The limit is path name-dependent.
- 9124 • The limit differs between the compile and runtime machines.

9125 For these reasons, an application may use the *fpathconf()*, *pathconf()*, and *sysconf()* functions to
 9126 determine the actual value of a limit at runtime.

9127 The items in the list ending in _MIN give the most negative values that the mathematical types
 9128 are guaranteed to be capable of representing. Numbers of a more negative value may be
 9129 supported on some implementations, as indicated by the <limits.h> header on the
 9130 implementation, but applications requiring such numbers are not guaranteed to be portable to
 9131 all implementations. For positive constants ending in _MIN, this indicates the minimum
 9132 acceptable value.

9133 The Minimum Acceptable Value symbol ' * ' indicates that there is no guaranteed value across
 9134 all conforming implementations.

9135 **Runtime Invariant Values (Possibly Indeterminate)**

9136 A definition of one of the symbolic names in the following list shall be omitted from <limits.h>
 9137 on specific implementations where the corresponding value is equal to or greater than the stated
 9138 minimum, but is indeterminate.

9139 This indetermination might depend on the amount of available memory space on a specific
 9140 instance of a specific implementation. The actual value supported by a specific instance shall be
 9141 provided by the *sysconf()* function.

9142 AIO {AIO_LISTIO_MAX}
 9143 Maximum number of I/O operations in a single list I/O call supported by the

9144		implementation.
9145		Minimum Acceptable Value: <code>{_POSIX_AIO_LISTIO_MAX}</code>
9146	AIO	<code>{AIO_MAX}</code>
9147		Maximum number of outstanding asynchronous I/O operations supported by the
9148		implementation.
9149		Minimum Acceptable Value: <code>{_POSIX_AIO_MAX}</code>
9150	AIO	<code>{AIO_PRIO_DELTA_MAX}</code>
9151		The maximum amount by which a process can decrease its asynchronous I/O priority level
9152		from its own scheduling priority.
9153		Minimum Acceptable Value: 0
9154		<code>{ARG_MAX}</code>
9155		Maximum length of argument to the <i>exec</i> functions including environment data.
9156		Minimum Acceptable Value: <code>{_POSIX_ARG_MAX}</code>
9157	XSI	<code>{ATEXIT_MAX}</code>
9158		Maximum number of functions that may be registered with <i>atexit()</i> .
9159		Minimum Acceptable Value: 32
9160		<code>{CHILD_MAX}</code>
9161		Maximum number of simultaneous processes per real user ID.
9162		Minimum Acceptable Value: 25
9163	TMR	<code>{DELAYTIMER_MAX}</code>
9164		Maximum number of timer expiration overruns.
9165		Minimum Acceptable Value: <code>{_POSIX_DELAYTIMER_MAX}</code>
9166	XSI	<code>{IOV_MAX}</code>
9167		Maximum number of <i>iovec</i> structures that one process has available for use with <i>readv()</i> or
9168		<i>writenv()</i> .
9169		Minimum Acceptable Value: <code>{_XOPEN_IOV_MAX}</code>
9170		<code>{LOGIN_NAME_MAX}</code>
9171		Maximum length of a login name.
9172		Minimum Acceptable Value: <code>{_POSIX_LOGIN_NAME_MAX}</code>
9173	MSG	<code>{MQ_OPEN_MAX}</code>
9174		The maximum number of open message queue descriptors a process may hold.
9175		Minimum Acceptable Value: <code>{_POSIX_MQ_OPEN_MAX}</code>
9176	MSG	<code>{MQ_PRIO_MAX}</code>
9177		The maximum number of message priorities supported by the implementation.
9178		Minimum Acceptable Value: <code>{_POSIX_MQ_PRIO_MAX}</code>
9179		<code>{OPEN_MAX}</code>
9180		Maximum number of files that one process can have open at any one time.
9181		Minimum Acceptable Value: 20
9182		<code>{PAGESIZE}</code>
9183		Size in bytes of a page.
9184		Minimum Acceptable Value: 1
9185	XSI	<code>{PAGE_SIZE}</code>
9186		Same as <code>{PAGESIZE}</code> . If either <code>{PAGESIZE}</code> or <code>{PAGE_SIZE}</code> is defined, the other is defined
9187		with the same value.

9188	THR	{PTHREAD_DESTRUCTOR_ITERATIONS}
9189		Maximum number of attempts made to destroy a thread's thread-specific data values on thread exit.
9190		
9191		Minimum Acceptable Value: {_POSIX_THREAD_DESTRUCTOR_ITERATIONS}
9192	THR	{PTHREAD_KEYS_MAX}
9193		Maximum number of data keys that can be created by a process.
9194		Minimum Acceptable Value: {_POSIX_THREAD_KEYS_MAX}
9195	THR	{PTHREAD_STACK_MIN}
9196		Minimum size in bytes of thread stack storage.
9197		Minimum Acceptable Value: 0
9198	THR	{PTHREAD_THREADS_MAX}
9199		Maximum number of threads that can be created per process.
9200		Minimum Acceptable Value: {_POSIX_THREAD_THREADS_MAX}
9201		{RE_DUP_MAX}
9202		The number of repeated occurrences of a BRE permitted by the <i>regex()</i> and <i>regcomp()</i> functions when using the interval notation $\{m,n\}$; see Section 9.3.6 (on page 201).
9203		
9204		Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}
9205	RTS	{RTSIG_MAX}
9206		Maximum number of realtime signals reserved for application use in this implementation.
9207		Minimum Acceptable Value: {_POSIX_RTSIG_MAX}
9208	SEM	{SEM_NSEMS_MAX}
9209		Maximum number of semaphores that a process may have.
9210		Minimum Acceptable Value: {_POSIX_SEM_NSEMS_MAX}
9211	SEM	{SEM_VALUE_MAX}
9212		The maximum value a semaphore may have.
9213		Minimum Acceptable Value: {_POSIX_SEM_VALUE_MAX}
9214	RTS	{SIGQUEUE_MAX}
9215		Maximum number of queued signals that a process may send and have pending at the receiver(s) at any time.
9216		
9217		Minimum Acceptable Value: {_POSIX_SIGQUEUE_MAX}
9218	SS TSP	{SS_REPL_MAX}
9219		The maximum number of replenishment operations that may be simultaneously pending for a particular sporadic server scheduler.
9220		
9221		Minimum Acceptable Value: {_POSIX_SS_REPL_MAX}
9222		{STREAM_MAX}
9223		The number of streams that one process can have open at one time. If defined, it has the same value as {FOPEN_MAX} (see <stdio.h>).
9224		
9225		Minimum Acceptable Value: {_POSIX_STREAM_MAX}
9226		{SYMLOOP_MAX}
9227		Maximum number of symbolic links that can be reliably traversed in the resolution of a path name in the absence of a loop.
9228		
9229		Minimum Acceptable Value: {_POSIX_SYMLOOP_MAX}
9230	TMR	{TIMER_MAX}
9231		Maximum number of timers per-process supported by the implementation.
9232		Minimum Acceptable Value: {_POSIX_TIMER_MAX}

9233 TRC {TRACE_EVENT_NAME_MAX}
 9234 Maximum length of the trace event name.
 9235 Minimum Acceptable Value: {_POSIX_TRACE_EVENT_NAME_MAX}

9236 TRC {TRACE_NAME_MAX}
 9237 Maximum length of the trace generation version string or of the trace stream name.
 9238 Minimum Acceptable Value: {_POSIX_TRACE_NAME_MAX}

9239 TRC {TRACE_SYS_MAX}
 9240 Maximum number of trace streams that may simultaneously exist in the system.
 9241 Minimum Acceptable Value: {_POSIX_TRACE_SYS_MAX}

9242 TRC {TRACE_USER_EVENT_MAX}
 9243 Maximum number of user trace event type identifiers that may simultaneously exist in a
 9244 traced process, including the predefined user trace event
 9245 _POSIX_TRACE_UNNAMED_USER_EVENT.
 9246 Minimum Acceptable Value: {_POSIX_TRACE_USER_EVENT_MAX}

9247 {TTY_NAME_MAX}
 9248 Maximum length of terminal device name.
 9249 Minimum Acceptable Value: {_POSIX_TTY_NAME_MAX}

9250 {TZNAME_MAX}
 9251 Maximum number of bytes supported for the name of a timezone (not of the TZ variable).
 9252 Minimum Acceptable Value: {_POSIX_TZNAME_MAX}

9253 **Note:** The length given by {TZNAME_MAX} does not include the quoting characters
 9254 mentioned in Section 8.3 (on page 192).

9255 **Path Name Variable Values**

9256 The values in the following list may be constants within an implementation or may vary from
 9257 one path name to another. For example, file systems or directories may have different
 9258 characteristics.

9259 A definition of one of the values shall be omitted from the <limits.h> header on specific
 9260 implementations where the corresponding value is equal to or greater than the stated minimum,
 9261 but where the value can vary depending on the file to which it is applied. The actual value
 9262 supported for a specific path name shall be provided by the *pathconf()* function.

9263 {FILESIZEBITS}
 9264 Minimum number of bits needed to represent, as a signed integer value, the maximum size
 9265 of a regular file allowed in the specified directory.
 9266 Minimum Acceptable Value: 32

9267 {LINK_MAX}
 9268 Maximum number of links to a single file.
 9269 Minimum Acceptable Value: {_POSIX_LINK_MAX}

9270 {MAX_CANON}
 9271 Maximum number of bytes in a terminal canonical input line.
 9272 Minimum Acceptable Value: {_POSIX_MAX_CANON}

9273 {MAX_INPUT}
 9274 Minimum number of bytes for which space is available in a terminal input queue; therefore,
 9275 the maximum number of bytes a portable application may require to be typed as input
 9276 before reading them.
 9277 Minimum Acceptable Value: {_POSIX_MAX_INPUT}

9278 {NAME_MAX}
 9279 Maximum number of bytes in a file name (not including terminating null).
 9280 Minimum Acceptable Value: {_POSIX_NAME_MAX}

9281 {PATH_MAX}
 9282 Maximum number of bytes in a path name, including the terminating null character.
 9283 Minimum Acceptable Value: {_POSIX_PATH_MAX}

9284 {PIPE_BUF}
 9285 Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
 9286 Minimum Acceptable Value: {_POSIX_PIPE_BUF}

9287 ADV {POSIX_ALLOC_SIZE_MIN}
 9288 Minimum number of bytes of storage actually allocated for any portion of a file.
 9289 Minimum Acceptable Value: Not specified.

9290 ADV {POSIX_REC_INCR_XFER_SIZE}
 9291 Recommended increment for file transfer sizes between the
 9292 {POSIX_REC_MIN_XFER_SIZE} and {POSIX_REC_MAX_XFER_SIZE} values.
 9293 Minimum Acceptable Value: Not specified.

9294 ADV {POSIX_REC_MAX_XFER_SIZE}
 9295 Maximum recommended file transfer size.
 9296 Minimum Acceptable Value: Not specified.

9297 ADV {POSIX_REC_MIN_XFER_SIZE}
 9298 Minimum recommended file transfer size.
 9299 Minimum Acceptable Value: Not specified.

9300 ADV {POSIX_REC_XFER_ALIGN}
 9301 Recommended file transfer buffer alignment.
 9302 Minimum Acceptable Value: Not specified.

9303 {SYMLINK_MAX}
 9304 Maximum number of bytes in a symbolic link.
 9305 Minimum Acceptable Value: {_POSIX_SYMLINK_MAX}

9306 **Runtime Inceasable Values**

9307 The magnitude limitations in the following list shall be fixed by specific implementations. An
 9308 application should assume that the value supplied by <limits.h> in a specific implementation is
 9309 the minimum that pertains whenever the application is run under that implementation. A
 9310 specific instance of a specific implementation may increase the value relative to that supplied by
 9311 <limits.h> for that implementation. The actual value supported by a specific instance shall be
 9312 provided by the *sysconf()* function.

9313 {BC_BASE_MAX}
 9314 Maximum *obase* values allowed by the *bc* utility.
 9315 Minimum Acceptable Value: {_POSIX2_BC_BASE_MAX}

9316 {BC_DIM_MAX}
 9317 Maximum number of elements permitted in an array by the *bc* utility.
 9318 Minimum Acceptable Value: {_POSIX2_BC_DIM_MAX}

9319 {BC_SCALE_MAX}
 9320 Maximum *scale* value allowed by the *bc* utility.
 9321 Minimum Acceptable Value: {_POSIX2_BC_SCALE_MAX}

- 9322 {BC_STRING_MAX}
- 9323 Maximum length of a string constant accepted by the *bc* utility.
- 9324 Minimum Acceptable Value: {_POSIX2_BC_STRING_MAX}

- 9325 {CHARCLASS_NAME_MAX}
- 9326 Maximum number of bytes in a character class name.
- 9327 Minimum Acceptable Value: {_POSIX2_CHARCLASS_NAME_MAX}

- 9328 {COLL_WEIGHTS_MAX}
- 9329 Maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order**
- 9330 keyword in the locale definition file; see Chapter 7 (on page 143).
- 9331 Minimum Acceptable Value: {_POSIX2_COLL_WEIGHTS_MAX}

- 9332 {EXPR_NEST_MAX}
- 9333 Maximum number of expressions that can be nested within parentheses by the *expr* utility.
- 9334 Minimum Acceptable Value: {_POSIX2_EXPR_NEST_MAX}

- 9335 {LINE_MAX}
- 9336 Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either
- 9337 standard input or another file), when the utility is described as processing text files. The
- 9338 length includes room for the trailing newline.
- 9339 Minimum Acceptable Value: {_POSIX2_LINE_MAX}

- 9340 {NGROUPS_MAX}
- 9341 Maximum number of simultaneous supplementary group IDs per process.
- 9342 Minimum Acceptable Value: 8

- 9343 {RE_DUP_MAX}
- 9344 Maximum number of repeated occurrences of a regular expression permitted when using
- 9345 the interval notation $\{m,n\}$; see Chapter 9 (on page 195).
- 9346 Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}

9347 **Maximum Values**

9348 TMR The symbolic constants in the following list shall be defined in <limits.h> with the values
9349 shown. These are symbolic names for the most restrictive value for certain features on an
9350 implementation supporting the Timers option. A conforming implementation shall provide
9351 values no larger than these values. A portable application must not require a smaller value for
9352 correct operation.

9353 TMR { _POSIX_CLOCKRES_MIN }

9354 The resolution of the CLOCK_REALTIME clock, in nanoseconds.

9355 Value: 20 000 000

9356 MON If the Monotonic Clock option is supported, the resolution of the CLOCK_MONOTONIC
9357 clock, in nanoseconds, is represented by { _POSIX_CLOCKRES_MIN }.

9358 **Minimum Values**

9359 The symbolic constants in the following list shall be defined in <limits.h> with the values
9360 shown. These are symbolic names for the most restrictive value for certain features on an
9361 implementation conforming to this volume of IEEE Std. 1003.1-200x. Related symbolic constants
9362 are defined elsewhere in this volume of IEEE Std. 1003.1-200x which reflect the actual
9363 implementation and which need not be as restrictive. A conforming implementation shall
9364 provide values at least this large. A strictly conforming application must not require a larger
9365 value for correct operation.

9366	AIO	{_POSIX_AIO_LISTIO_MAX}
9367		The number of I/O operations that can be specified in a list I/O call.
9368		Value: 2
9369	AIO	{_POSIX_AIO_MAX}
9370		The number of outstanding asynchronous I/O operations.
9371		Value: 1
9372		{_POSIX_ARG_MAX}
9373		Maximum length of argument to the <i>exec</i> functions including environment data.
9374		Value: 4 096
9375		{_POSIX_CHILD_MAX}
9376		Maximum number of simultaneous processes per real user ID.
9377		Value: 6
9378	TMR	{_POSIX_DELAYTIMER_MAX}
9379		The number of timer expiration overruns.
9380		Value: 32
9381		{_POSIX_LINK_MAX}
9382		Maximum number of links to a single file.
9383		Value: 8
9384		{_POSIX_LOGIN_NAME_MAX}
9385		The size of the storage required for a login name, in bytes, including the terminating null.
9386		Value: 9
9387		{_POSIX_MAX_CANON}
9388		Maximum number of bytes in a terminal canonical input queue.
9389		Value: 255
9390		{_POSIX_MAX_INPUT}
9391		Maximum number of bytes allowed in a terminal input queue.
9392		Value: 255
9393	MSG	{_POSIX_MQ_OPEN_MAX}
9394		The number of message queues that can be open for a single process.
9395		Value: 8
9396	MSG	{_POSIX_MQ_PRIO_MAX}
9397		The maximum number of message priorities supported by the implementation.
9398		Value: 32

9399 **Notes to Reviewers**

9400 *This section with side shading will not appear in the final copy. - Ed.*

9401 D1, XSH, ERN 436 proposes increasing the value of {_POSIX_NAME_MAX} to 256.

9402 Similarly, it proposes {_POSIX_PATH_MAX} be 1 024.

9403 {_POSIX_NAME_MAX}

9404 Maximum number of bytes in a file name (not including terminating null).

9405 Value: 14

9406 **Notes to Reviewers**

9407 *This section with side shading will not appear in the final copy. - Ed.*

9408 D1, XSH, ERN 19 proposes to increase `{_POSIX_NGROUPS_MAX}`, `{_POSIX_OPEN_MAX}`,
9409 and `{_POSIX_CHILD_MAX}` to their FIPS values (8, 20, 25) as with the limits equivalents
9410 without the leading `_POSIX`).

9411 `{_POSIX_NGROUPS_MAX}`
9412 Maximum number of simultaneous supplementary group IDs per process.
9413 Value: 0

9414 `{_POSIX_OPEN_MAX}`
9415 Maximum number of files that one process can have open at any one time.
9416 Value: 16

9417 `{_POSIX_PATH_MAX}`
9418 Maximum number of bytes in a path name.
9419 Value: 256

9420 `{_POSIX_PIPE_BUF}`
9421 Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
9422 Value: 512

9423 `{_POSIX_RE_DUP_MAX}`
9424 The number of repeated occurrences of a BRE permitted by the `regex()` and `regcomp()`
9425 functions when using the interval notation `{\ (m,n)\ }`; see Section 9.3.6 (on page 201).
9426 Value: 255

9427 RTS `{_POSIX_RTSIG_MAX}`
9428 The number of realtime signal numbers reserved for application use.
9429 Value: 8

9430 SEM `{_POSIX_SEM_NSEMS_MAX}`
9431 The number of semaphores that a process may have.
9432 Value: 256

9433 SEM `{_POSIX_SEM_VALUE_MAX}`
9434 The maximum value a semaphore may have.
9435 Value: 32 767

9436 RTS `{_POSIX_SIGQUEUE_MAX}`
9437 The number of queued signals that a process may send and have pending at the receiver(s)
9438 at any time.
9439 Value: 32

9440 `{_POSIX_SSIZE_MAX}`
9441 The value that can be stored in an object of type `ssize_t`.
9442 Value: 32 767

9443 `{_POSIX_STREAM_MAX}`
9444 The number of streams that one process can have open at one time.
9445 Value: 8

9446 SS|TSP `{_POSIX_SS_REPL_MAX}`
9447 The number of replenishment operations that may be simultaneously pending for a
9448 particular sporadic server scheduler.
9449 Value: 4

9450 {_POSIX_SYMLINK_MAX}
 9451 The number of bytes in a symbolic link.
 9452 Value: 255

9453 {_POSIX_SYMLOOP_MAX}
 9454 The number of symbolic links that can be traversed in the resolution of a path name in the
 9455 absence of a loop.
 9456 Value: 8

9457 THR {_POSIX_THREAD_DESTRUCTOR_ITERATIONS}
 9458 The number of attempts made to destroy a thread's thread-specific data values on thread
 9459 exit.
 9460 Value: 4

9461 THR {_POSIX_THREAD_KEYS_MAX}
 9462 The number of data keys per process.
 9463 Value: 128

9464 THR {_POSIX_THREAD_THREADS_MAX}
 9465 The number of threads per process.
 9466 Value: 64

9467 TMR {_POSIX_TIMER_MAX}
 9468 The per process number of timers.
 9469 Value: 32

9470 TRC {_POSIX_TRACE_EVENT_NAME_MAX}
 9471 The length in bytes of a trace event name.
 9472 Value: 30

9473 TRC {_POSIX_TRACE_NAME_MAX}
 9474 The length in bytes of a trace generation version string or a trace stream name.
 9475 Value: 8

9476 TRC {_POSIX_TRACE_SYS_MAX}
 9477 The number of trace streams that may simultaneously exist in the system.
 9478 Value: 8

9479 TRC {_POSIX_TRACE_USER_EVENT_MAX}
 9480 The number of user trace event type identifiers that may simultaneously exist in a traced
 9481 process, including the predefined user trace event
 9482 _POSIX_TRACE_UNNAMED_USER_EVENT.
 9483 Value: 32

9484 {_POSIX_TTY_NAME_MAX}
 9485 The size of the storage required for a terminal device name, in bytes, including the
 9486 terminating null.
 9487 Value: 9

9488 {_POSIX_TZNAME_MAX}
 9489 Maximum number of bytes supported for the name of a timezone (not of the TZ variable).
 9490 Value: 6

9491 **Note:** The length given by {_POSIX_TZNAME_MAX} does not include the quoting
 9492 characters mentioned in Section 8.3 (on page 192).

9493 {_POSIX2_BC_BASE_MAX}
 9494 Maximum *obase* values allowed by the *bc* utility.
 9495 Value: 99

9496 { _POSIX2_BC_DIM_MAX}
 9497 Maximum number of elements permitted in an array by the *bc* utility.
 9498 Value: 2 048

9499 { _POSIX2_BC_SCALE_MAX}
 9500 Maximum *scale* value allowed by the *bc* utility.
 9501 Value: 99

9502 { _POSIX2_BC_STRING_MAX}
 9503 Maximum length of a string constant accepted by the *bc* utility.
 9504 Value: 1 000

9505 { _POSIX2_CHARCLASS_NAME_MAX}
 9506 Maximum number of bytes in a character class name.
 9507 Value: 14

9508 { _POSIX2_COLL_WEIGHTS_MAX}
 9509 Maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order**
 9510 keyword in the locale definition file; see Chapter 7 (on page 143).
 9511 Value: 2

9512 { _POSIX2_EXPR_NEST_MAX}
 9513 Maximum number of expressions that can be nested within parentheses by the *expr* utility.
 9514 Value: 32

9515 { _POSIX2_LINE_MAX}
 9516 Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either
 9517 standard input or another file), when the utility is described as processing text files. The
 9518 length includes room for the trailing newline.
 9519 Value: 2 048

9520 { _POSIX2_RE_DUP_MAX}
 9521 Maximum number of repeated occurrences of a regular expression permitted when using
 9522 the interval notation $\{m,n\}$; see Chapter 9 (on page 195).
 9523 Value: 255

9524 XSI { _XOPEN_IOV_MAX}
 9525 Maximum number of **iovec** structures that one process has available for use with *readv()* or
 9526 *writenv()*.
 9527 Value: 16
 9528

9529 Numerical Limits

9530 The values in the following lists shall be defined in **<limits.h>** and are constant expressions
 9531 XSI suitable for use in **#if** preprocessing directives. Moreover, except for {CHAR_BIT}, {DBL_DIG},
 9532 {DBL_MAX}, {FLT_DIG}, {FLT_MAX}, {LONG_BIT}, {WORD_BIT}, and {MB_LEN_MAX}, the
 9533 symbolic names are defined as expressions of the correct type.

9534 If the value of an object of type **char** is treated as a signed integer when used in an expression,
 9535 the value of {CHAR_MIN} is the same as that of {SCHAR_MIN} and the value of {CHAR_MAX}
 9536 is the same as that of {SCHAR_MAX}. Otherwise, the value of {CHAR_MIN} is 0 and the value
 9537 of {CHAR_MAX} is the same as that of {UCHAR_MAX}.

9538 {CHAR_BIT}
 9539 Number of bits in a type **char**.
 9540 Minimum Acceptable Value: 8

9541 {CHAR_MAX}
 9542 Maximum value of type **char**.
 9543 Minimum Acceptable Value: {UCHAR_MAX} or {SCHAR_MAX}

9544 {INT_MAX}
 9545 Maximum value of an **int**.
 9546 Minimum Acceptable Value: 2 147 483 647

9547 XSI {LONG_BIT}
 9548 Number of bits in a **long**.
 9549 Minimum Acceptable Value: 32

9550 {LONG_MAX}
 9551 Maximum value of a **long**.
 9552 Minimum Acceptable Value: +2 147 483 647

9553 {MB_LEN_MAX}
 9554 Maximum number of bytes in a character, for any supported locale.
 9555 Minimum Acceptable Value: 1

9556 {SCHAR_MAX}
 9557 Maximum value of type **signed char**.
 9558 Minimum Acceptable Value: +127

9559 {SHRT_MAX}
 9560 Maximum value of type **short**.
 9561 Minimum Acceptable Value: +32 767

9562 {SSIZE_MAX}
 9563 Maximum value of an object of type **ssize_t**.
 9564 Minimum Acceptable Value: {_POSIX_SSIZE_MAX}

9565 {UCHAR_MAX}
 9566 Maximum value of type **unsigned char**.
 9567 Minimum Acceptable Value: 255

9568 {UINT_MAX}
 9569 Maximum value of type **unsigned**.
 9570 Minimum Acceptable Value: 4 294 967 295

9571 {ULONG_MAX}
 9572 Maximum value of type **unsigned long**.
 9573 Minimum Acceptable Value: 4 294 967 295

9574 {USHRT_MAX}
 9575 Maximum value for a type **unsigned short**.
 9576 Minimum Acceptable Value: 65 535

9577 XSI {WORD_BIT}
 9578 Number of bits in a word or type **int**.
 9579 Minimum Acceptable Value: 16

9580 {CHAR_MIN}
 9581 Minimum value of type **char**.
 9582 Maximum Acceptable Value: {SCHAR_MIN} or 0

9583 {INT_MIN}
 9584 Minimum value of type **int**.
 9585 Maximum Acceptable Value: -2 147 483 647

9586 {LONG_MIN}
 9587 Minimum value of type **long**.
 9588 Maximum Acceptable Value: -2 147 483 647

9589 {SCHAR_MIN}
 9590 Minimum value of type **signed char**.
 9591 Maximum Acceptable Value: -127

9592 {SHRT_MIN}
 9593 Minimum value of type **short**.
 9594 Maximum Acceptable Value: -32 767

9595 {LLONG_MIN}
 9596 Minimum value of type **long long**.
 9597 Maximum Acceptable Value: -9223372036854775807

9598 {LLONG_MAX}
 9599 Maximum value of type **long long**.
 9600 Minimum Acceptable Value: +9223372036854775807

9601 {ULLONG_MAX}
 9602 Maximum value of type **unsigned long long**.
 9603 Minimum Acceptable Value: 18446744073709551615

9604 **Other Invariant Values**

9605 XSI The following constants shall be defined on all implementations in <limits.h>:

9606 XSI {CHARCLASS_NAME_MAX}
 9607 Maximum number of bytes in a character class name.
 9608 Minimum Acceptable Value: 14

9609 XSI {NL_ARGMAX}
 9610 Maximum value of *digit* in calls to the *printf()* and *scanf()* functions.
 9611 Minimum Acceptable Value: 9

9612 XSI {NL_LANGMAX}
 9613 Maximum number of bytes in a *LANG* name.
 9614 Minimum Acceptable Value: 14

9615 XSI {NL_MSGMAX}
 9616 Maximum message number.
 9617 Minimum Acceptable Value: 32 767

9618 XSI {NL_NMAX}
 9619 Maximum number of bytes in an N-to-1 collation mapping.
 9620 Minimum Acceptable Value: ' * '

9621 XSI {NL_SETMAX}
 9622 Maximum set number.
 9623 Minimum Acceptable Value: 255

9624 XSI {NL_TEXTMAX}
 9625 Maximum number of bytes in a message string.
 9626 Minimum Acceptable Value: {_POSIX2_LINE_MAX}

9627 XSI {NZERO}
 9628 Default process priority.
 9629 Minimum Acceptable Value: 20

9630 XSI {TMP_MAX}
 9631 Minimum number of unique path names generated by *tmpnam()*. Maximum number of
 9632 times an application can call *tmpnam()* reliably. (LEGACY)
 9633 Minimum Acceptable Value: 10 000

9634 **APPLICATION USAGE**
 9635 None.

9636 **RATIONALE**
 9637 A request was made to reduce the value of {_POSIX_LINK_MAX} from the value of 8 specified
 9638 for it in the POSIX.1-1990 standard to 2. The standard developers decided to deny this request
 9639 for several reasons.

- 9640 • They wanted to avoid making any changes to the standard that could break conforming
 9641 applications, and the requested change could have that effect.
- 9642 • The use of multiple hard links to a file cannot always be replaced with use of symbolic links.
 9643 Symbolic links are semantically different from hard links in that they associate a path name
 9644 with another path name rather than a path name with a file. This has implications for access
 9645 control, file permanence, and transparency.
- 9646 • The original standard developers had considered the issue of allowing for implementations
 9647 that did not in general support hard links, and decided that this would reduce consensus on
 9648 the standard.

9649 Systems that support historical versions of the development option of the ISO POSIX-2 standard
 9650 retain the name {_POSIX2_RE_DUP_MAX} as an alias for {_POSIX_RE_DUP_MAX}.

9651 {PATH_MAX}
 9652 IEEE PASC Interpretation 1003.1 #15 addressed the inconsistency in the standard with the
 9653 definition of path name and the description of {PATH_MAX}, allowing application writers
 9654 to allocate either {PATH_MAX} or {PATH_MAX}+1 bytes. The inconsistency has been
 9655 removed by correction to the {PATH_MAX} definition to include the null character. With
 9656 this change, applications that previously allocated {PATH_MAX} bytes will continue to
 9657 succeed.

9658 {SYMLINK_MAX}
 9659 This symbol refers to space for data that is stored in the file system, as opposed to
 9660 {PATH_MAX} which is the length of a name that can be passed to a function. In some
 9661 existing implementations, the file names pointed to by symbolic links are stored in the
 9662 inodes of the links, so it is important that {SYMLINK_MAX} not be constrained to be as
 9663 large as {PATH_MAX}.

9664 **FUTURE DIRECTIONS**
 9665 None.

9666 **SEE ALSO**
 9667 The System Interfaces volume of IEEE Std. 1003.1-200x, *fpathconf()*, *pathconf()*, *sysconf()*

9668 **CHANGE HISTORY**
 9669 First released in Issue 1.

9670 **Issue 4**
 9671 A sentence is added to the DESCRIPTION indicating that names beginning with _POSIX can be
 9672 found in <unistd.h>.

9673 The {PASS_MAX} and {TMP_MAX} symbols are marked LEGACY.

9674 Use of the terms “bytes” and “characters” is rationalized to make it clear when the description is
9675 referring to either single-byte values or possibly multi-byte characters.

9676 {CHARCLASS_NAME_MAX} is added to the list of **Other Invariant Values** and marked as an
9677 extension.

9678 This entry is largely restructured to improve symbol grouping. A great many symbols, too
9679 numerous to mention, have also been added for alignment with the ISO POSIX-2 standard.

9680 The following changes are incorporated for alignment with the ISO C standard:

9681 • The constants {INT_MIN}, {LONG_MIN}, and {SHRT_MIN} are changed from values ending
9682 in 8 to ones ending in 7.

9683 • The {DBL_DIG}, {DBL_MAX}, {FLT_DIG}, and {FLT_MAX} symbols are marked both as
9684 extensions and LEGACY.

9685 • The {LONG_BIT} and {WORD_BIT} symbols are marked as extensions.

9686 • The {DBL_MIN} and {FLT_MIN} symbols are withdrawn.

9687 • Text introducing numerical limits now indicates that they are constant expressions suitable
9688 for use in #if preprocessing directives.

9689 The following change is incorporated for alignment with the FIPS requirements:

9690 • The minimum acceptable value for {NGROUPS_MAX} is changed from
9691 {_POSIX_NGROUPS_MAX} to 8. This is marked as an extension.

9692 **Issue 4, Version 2**

9693 The DESCRIPTION is revised for X/OPEN UNIX conformance as follows:

9694 • Under **Runtime Invariant Values**, {ATEXIT_MAX}, {IOV_MAX}, {PAGESIZE}, and
9695 {PAGE_SIZE} are added.

9696 • Under **Minimum Values**, {_XOPEN_IOV_MAX} is added.

9697 **Issue 5**

9698 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
9699 Threads Extension.

9700 {FILESIZEBITS} added for the Large File Summit extensions.

9701 The minimum acceptable values for {INT_MAX}, {INT_MIN}, and {UINT_MAX} are changed to
9702 make 32-bit values the minimum requirement.

9703 The entry is restructured to improve readability.

9704 **Issue 6**

9705 The Open Group corrigenda item U033/4 has been applied. The wording is made clear for
9706 {CHAR_MIN}, {INT_MIN}, {LONG_MIN}, {SCHAR_MIN}, and {SHRT_MIN} that these are
9707 maximum acceptable values.

9708 The following new requirements on POSIX implementations derive from alignment with the
9709 Single UNIX Specification:

9710 • The minimum value for {CHILD_MAX} is 25. This is a FIPS requirement.

9711 • The minimum value for {OPEN_MAX} is 20. This is a FIPS requirement.

9712 • The minimum value for {NGROUPS_MAX} is 8. This is also a FIPS requirement.

9713 Symbolic constants are added for {_POSIX_SYMLINK_MAX}, {_POSIX_SYMLOOP_MAX},
9714 {_POSIX_RE_DUP_MAX}, {RE_DUP_MAX}, {SYMLOOP_MAX}, and {SYMLINK_MAX}.

9715 The following values are added for alignment with IEEE Std. 1003.1d-1999:

9716 {_POSIX_SS_REPL_MAX}
 9717 {SS_REPL_MAX}
 9718 {POSIX_ALLOC_SIZE_MIN}
 9719 {POSIX_REC_INCR_XFER_SIZE}
 9720 {POSIX_REC_MAX_XFER_SIZE}
 9721 {POSIX_REC_MIN_XFER_SIZE}
 9722 {POSIX_REC_XFER_ALIGN}

9723 Reference to CLOCK_MONOTONIC is added in the description of {_POSIX_CLOCKRES_MIN}
 9724 for alignment with IEEE Std. 1003.1j-2000.

9725 The constants {LLONG_MIN}, {LLONG_MAX}, and {ULLONG_MAX} are added for alignment
 9726 with the ISO/IEC 9899:1999 standard.

9727 The following values are added for alignment with IEEE Std. 1003.1q-2000:
 9728 {_POSIX_TRACE_EVENT_NAME_MAX}, {_POSIX_TRACE_NAME_MAX},
 9729 {_POSIX_TRACE_SYS_MAX}, {_POSIX_TRACE_USER_EVENT_MAX},
 9730 {TRACE_EVENT_NAME_MAX}, {TRACE_NAME_MAX}, {TRACE_SYS_MAX},
 9731 {TRACE_USER_EVENT_MAX}

9732 **NAME**

9733 locale.h — category macros

9734 **SYNOPSIS**

9735 #include <locale.h>

9736 **DESCRIPTION**

9737 cx The functionality described on this reference page extends the ISO C standard. Applications
 9738 shall define the appropriate feature test macro (see the System Interfaces volume of
 9739 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 9740 symbols in this header.

9741 The <locale.h> header shall provide a definition for structure **lconv**, which shall include at least
 9742 the following members. (See the definitions of *LC_MONETARY* in the Section 7.3.3 (on page
 9743 163), and Section 7.3.4 (on page 166).)

9744 char *currency_symbol
 9745 char *decimal_point
 9746 char frac_digits
 9747 char *grouping
 9748 char *int_curr_symbol
 9749 char int_frac_digits
 9750 char int_n_cs_precedes
 9751 char int_n_sep_by_space
 9752 char int_n_sign_posn
 9753 char int_p_cs_precedes
 9754 char int_p_sep_by_space
 9755 char int_p_sign_posn
 9756 char *mon_decimal_point
 9757 char *mon_grouping
 9758 char *mon_thousands_sep
 9759 char *negative_sign
 9760 char n_cs_precedes
 9761 char n_sep_by_space
 9762 char n_sign_posn
 9763 char *positive_sign
 9764 char p_cs_precedes
 9765 char p_sep_by_space
 9766 char p_sign_posn
 9767 char *thousands_sep

9768 The <locale.h> header shall define NULL (as defined in <stddef.h>) and at least the following as
 9769 macros:

9770 *LC_ALL*
 9771 *LC_COLLATE*
 9772 *LC_CTYPE*
 9773 *LC_MESSAGES*
 9774 *LC_MONETARY*
 9775 *LC_NUMERIC*
 9776 *LC_TIME*

9777 which shall expand to distinct integral constant expressions, for use as the first argument to the
 9778 *setlocale()* function.

9779 Additional macro definitions, beginning with the characters *LC_* and an uppercase letter, may
9780 also be given here.

9781 The following shall be declared as functions and may also be defined as macros. Function
9782 prototypes shall be provided for use with an ISO C standard compiler.

```
9783 struct lconv *localeconv (void);  
9784 char    setlocale(int, const char *);
```

9785 APPLICATION USAGE

9786 None.

9787 RATIONALE

9788 None.

9789 FUTURE DIRECTIONS

9790 None.

9791 SEE ALSO

9792 The System Interfaces volume of IEEE Std. 1003.1-200x, *localeconv()*, *setlocale()*, Chapter 8 (on
9793 page 187)

9794 CHANGE HISTORY

9795 First released in Issue 3.

9796 Entry included for alignment with the ISO C standard.

9797 Issue 4

9798 The following changes are incorporated for alignment with the ISO C standard:

- 9799 • The function declarations in this header are expanded to full ISO C standard prototypes.
- 9800 • The definition of **struct lconv** is added.
- 9801 • A reference to <stddef.h> is added for the definition of NULL.

9802 Issue 6

9803 The **lconv** structure is expanded with new members (**int_n_cs_precedes**, **int_n_sep_by_space**,
9804 **int_n_sign_posn**, **int_p_cs_precedes**, **int_p_sep_by_space**, and **int_p_sign_posn**) for alignment
9805 with the ISO/IEC 9899:1999 standard.

9806 **NAME**9807 `math.h` — mathematical declarations9808 **SYNOPSIS**9809 `#include <math.h>`9810 **DESCRIPTION**

9811 `CX` The functionality described on this reference page extends the ISO C standard. Applications
 9812 shall define the appropriate feature test macro (see the System Interfaces volume of
 9813 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 9814 symbols in this header.

9815 The **<math.h>** header shall include definitions for at least the following types:

9816 **float_t** A floating type at least as wide as **float**.

9817 **double_t** A floating type at least as wide as **double**, and at least as wide as **float_t**.

9818 If `FLT_EVAL_METHOD` equals 0, **float_t** and **double_t** shall be **float** and **double**, respectively; if
 9819 `FLT_EVAL_METHOD` equals 1, they shall both be **double**; if `FLT_EVAL_METHOD` equals 2,
 9820 they shall both be **long double**; for other values of `FLT_EVAL_METHOD`, they are otherwise
 9821 implementation-defined.

9822 The **<math.h>** header shall define the following macros, where **real-floating** indicates that the
 9823 argument shall be an expression of **real-floating** type:

```
9824 int fpclassify(real-floating x);
9825 int isfinite(real-floating x);
9826 int isinf(real-floating x);
9827 int isnan(real-floating x);
9828 int isnormal(real-floating x);
9829 int signbit(real-floating x);
9830 int isgreater(real-floating x, real-floating y);
9831 int isgreaterequal(real-floating x, real-floating y);
9832 int isless(real-floating x, real-floating y);
9833 int islessequal(real-floating x, real-floating y);
9834 int islessgreater(real-floating x, real-floating y);
9835 int isunordered(real-floating x, real-floating y);
```

9836 The **<math.h>** header shall provide for the following constants. The values are of type **double**
 9837 and are accurate within the precision of the **double** type.

9838	<code>XSI</code>	<code>M_E</code>	Value of e
9839		<code>M_LOG2E</code>	Value of $\log_2 e$
9840		<code>M_LOG10E</code>	Value of $\log_{10} e$
9841		<code>M_LN2</code>	Value of $\log_e 2$
9842		<code>M_LN10</code>	Value of $\log_e 10$
9843		<code>M_PI</code>	Value of π
9844		<code>M_PI_2</code>	Value of $\pi/2$
9845		<code>M_PI_4</code>	Value of $\pi/4$
9846		<code>M_1_PI</code>	Value of $1/\pi$
9847		<code>M_2_PI</code>	Value of $2/\pi$

9848	M_2_SQRTPI	Value of $2\sqrt{\pi}$
9849	M_SQRT2	Value of $\sqrt{2}$
9850	M_SQRT1_2	Value of $1\sqrt{2}$
9851	The header shall define the following symbolic constants:	
9852	XSI MAXFLOAT	Value of maximum non-infinite single-precision floating-point number.
9853	HUGE_VAL	A positive double expression, not necessarily representable as a float . Used as an error value returned by the mathematics library. HUGE_VAL evaluates to $+\infty$ on systems supporting IEEE Std. 754-1985.
9854		
9855		
9856	HUGE_VALF	A positive float constant expression. Used as an error value returned by the mathematics library. HUGE_VALF evaluates to $+\infty$ on systems supporting IEEE Std. 754-1985.
9857		
9858		
9859	HUGE_VALD	A positive long double constant expression. Used as an error value returned by the mathematics library. HUGE_VALD evaluates to $+\infty$ on systems supporting IEEE Std. 754-1985.
9860		
9861		
9862	INFINITY	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.
9863		
9864		
9865	NAN	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.
9866		
9867		
9868	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 FP_ and an uppercase letter, may also be specified by the implementation.	
9869		
9870		
9871		
9872		
9873	FP_INFINITE	
9874	FP_NAN	
9875	FP_NORMAL	
9876	FP_SUBNORMAL	
9877	FP_ZERO	
9878	The following macros are optional. If FP_FAST_FMA is defined, it shall indicate that the <i>fma()</i> function generally executes about as fast as, or faster than, a multiply and an add of double operands.	
9879		
9880		
9881	FP_FAST_FMA	
9882	FP_FAST_FMAF	
9883	FP_FAST_FMAL	
9884	FP_FAST_FMAF and FP_FAST_FMAL are, respectively, float and long double analogs of FP_FAST_FMA.	
9885		
9886	The following macros shall expand to integer constant expressions whose values are returned by <i>ilogb(x)</i> if <i>x</i> is zero or NaN, respectively. The value of FP_ILOGB0 shall be either {INT_MIN} or $-\{INT_MAX\}$. The value of FP_ILOGBNAN shall be either {INT_MAX} or {INT_MIN}.	
9887		
9888		
9889	FP_ILOGB0	
9890	FP_ILOGBNAN	

9891 The following macros shall expand to the integer constants 1 and 2, respectively;

9892 MATH_ERRNO
9893 MATH_ERREXCEPT

9894 The following macro shall expand to an expression that has type **int** and the value
9895 MATH_ERRNO, MATH_ERREXCEPT, or the bitwise-inclusive OR of both. The value of
9896 *math_errhandling* is constant for the duration of the program. It is unspecified whether
9897 *math_errhandling* is a macro or an identifier with external linkage. If a macro definition is
9898 suppressed or a program defines an identifier with the name *math_errhandling*, the behavior is
9899 undefined. If the expression *math_errhandling* & MATH_ERREXCEPT can be non-zero, the
9900 implementation shall define the macros FE_DIVBYZERO, FE_INVALID, and FE_OVERFLOW in
9901 **<fenv.h>**.

9902 *math_errhandling*

9903 The following shall be declared as functions and may also be defined as macros. Function
9904 prototypes shall be provided for use with an ISO C standard compiler.

9905 double acos(double);
9906 float acosf(float);
9907 XSI double acosh(double);
9908 float acoshf(float);
9909 long double acoshl(long double);
9910 long double acosl(long double);
9911 double asin(double);
9912 float asinf(float);
9913 XSI double asinh(double);
9914 float asinhf(float);
9915 long double asinhl(long double);
9916 long double asinl(long double);
9917 double atan(double);
9918 double atan2(double, double);
9919 float atan2f(float, float);
9920 long double atan2l(long double, long double);
9921 float atanf(float);
9922 XSI double atanh(double);
9923 float atanhf(float);
9924 long double atanh1(long double);
9925 long double atanl(long double);
9926 XSI double cbrt(double);
9927 float cbrtf(float);
9928 long double cbrtl(long double);
9929 double ceil(double);
9930 float ceilf(float);
9931 long double ceill(long double);
9932 double copysign(double, double);
9933 float copysignf(float, float);
9934 long double copysignl(long double, long double);
9935 double cos(double);
9936 float cosf(float);
9937 double cosh(double);
9938 float coshf(float);
9939 long double coshl(long double);
9940 long double cosl(long double);

```

9941 XSI    double    erf(double);
9942        double    erfc(double);
9943        float    erfcf(float);
9944        long double    erfcl(long double);
9945        float    erff(float);
9946        long double    erfl(long double);
9947        double    exp(double);
9948        double    exp2(double);
9949        float    exp2f(float);
9950        long double    exp2l(long double);
9951        float    expf(float);
9952        long double    expl(long double);
9953 XSI    double    expm1(double);
9954        float    expm1f(float);
9955        long double    expm1l(long double);
9956        double    fabs(double);
9957        float    fabsf(float);
9958        long double    fabsl(long double);
9959        double    fdim(double, double);
9960        float    fdimf(float, float);
9961        long double    fdiml(long double, long double);
9962        double    floor(double);
9963        float    floorf(float);
9964        long double    floorl(long double);
9965        double    fma(double, double, double);
9966        float    fmaf(float, float, float);
9967        long double    fmal(long double, long double, long double);
9968        double    fmax(double, double);
9969        float    fmaxf(float, float);
9970        long double    fmaxl(long double, long double);
9971        double    fmin(double, double);
9972        float    fminf(float, float);
9973        long double    fminl(long double, long double);
9974        double    fmod(double, double);
9975        float    fmodf(float, float);
9976        long double    fmodl(long double, long double);
9977        double    frexp(double, int *);
9978        float    frexpf(float value, int *);
9979        long double    frexpl(long double value, int *);
9980 XSI    double    hypot(double, double);
9981        float    hypotf(float, float);
9982        long double    hypotl(long double, long double);
9983 XSI    int    ilogb(double);
9984        int    ilogbf(float);
9985        int    ilogbl(long double);
9986 XSI    int    isnan(double);
9987        double    j0(double);
9988        double    j1(double);
9989        double    jn(int, double);
9990        double    ldexp(double, int);
9991        float    ldexpf(float, int);
9992        long double    ldexpl(long double, int);

```

```
9993 XSI    double    lgamma(double);
9994        float    lgammaf(float);
9995        long double lgammal(long double);
9996        double    log(double);
9997        double    log10(double);
9998        float    log10f(float);
9999        long double log10l(long double);
10000 XSI   double    log1p(double);
10001        float    log1pf(float);
10002        long double log1pl(long double);
10003        double    log2(double);
10004        float    log2f(float);
10005        long double log2l(long double);
10006 XSI   double    logb(double);
10007        float    logbf(float);
10008        long double logbl(long double);
10009        float    logf(float);
10010        long double logl(long double);
10011        long long llrint(double);
10012        long long llrintf(float);
10013        long long llrintl(long double);
10014        long long llround(double);
10015        long long llroundf(float);
10016        long long llroundl(long double);
10017        long    lrint(double);
10018        long    lrintf(float);
10019        long    lrintl(long double);
10020        long    lround(double);
10021        long    lroundf(float);
10022        long    lroundl(long double);
10023        double   modf(double, double *);
10024        float   modff(float, float *);
10025        long double modfl(long double, long double *);
10026        double   nan(const char *);
10027        float   nanf(const char *);
10028        long double nanl(const char *);
10029        double   nearbyint(double);
10030        float   nearbyintf(float);
10031        long double nearbyintl(long double);
10032 XSI   double   nextafter(double, double);
10033        float   nextafterf(float, float);
10034        long double nextafterl(long double, long double);
10035        double   nexttoward(double, long double);
10036        float   nexttowardf(float, long double);
10037        long double nexttowardl(long double, long double);
10038        double   pow(double, double);
10039        float   powf(float, float);
10040        long double powl(long double, long double);
10041 XSI   double   remainder(double, double);
10042        float   remainderf(float, float);
10043        long double remainderl(long double, long double);
10044        double   remquo(double, double, int *);
```

```

10045     float      remquof(float, float, int *);
10046     long double remquol(long double, long double, int *);
10047 XSI   double      rint(double);
10048     float      rintf(float);
10049     long double rintl(long double);
10050     double     round(double);
10051     float      roundf(float);
10052     long double roundl(long double);
10053 XSI   double     scalb(double, double);
10054     double     scalbln(double, long);
10055     float      scalblnf(float, long);
10056     long double scalblnl(long double, long);
10057     double     scalbn(double, int);
10058     float      scalbnf(float, int);
10059     long double scalbnl(long double, int);
10060     double     sin(double);
10061     float      sinf(float);
10062     double     sinh(double);
10063     float      sinhf(float);
10064     long double sinhl(long double);
10065     long double sinl(long double);
10066     double     sqrt(double);
10067     float      sqrtf(float);
10068     long double sqrtl(long double);
10069     double     tan(double);
10070     float      tanf(float);
10071     double     tanh(double);
10072     float      tanhf(float);
10073     long double tanhl(long double);
10074     long double tanl(long double);
10075     double     tgamma(double);
10076     float      tgammaf(float);
10077     long double tgamma_l(long double);
10078     double     trunc(double);
10079     float      truncf(float);
10080     long double trunc_l(long double);
10081 XSI   double     y0(double);
10082     double     y1(double);
10083     double     yn(int, double);
10084

```

10085 **The following external variable shall be defined:**

```

10086 XSI   extern int signgam;
10087

```

10088 APPLICATION USAGE

10089 The FP_CONTRACT pragma can be used to allow (if the state is on) or disallow (if the state is
10090 off) the implementation to contract expressions. Each pragma can occur either outside external
10091 declarations or preceding all explicit declarations and statements inside a compound statement.
10092 When outside external declarations, the pragma takes effect from its occurrence until another
10093 FP_CONTRACT pragma is encountered, or until the end of the translation unit. When inside a
10094 compound statement, the pragma takes effect from its occurrence until another FP_CONTRACT
10095 pragma is encountered (including within a nested compound statement), or until the end of the
10096 compound statement; at the end of a compound statement the state for the pragma is restored to
10097 its condition just before the compound statement. If this pragma is used in any other context, the
10098 behavior is undefined. The default state (on or off) for the pragma is implementation-defined.

10099 RATIONALE

10100 Before the ISO/IEC 9899:1999 standard, the math library was defined only for the floating type
10101 **double**. All the names formed by appending 'f' or 'l' to a name in **<math.h>** were reserved
10102 to allow for the definition of **float** and **long double** libraries; and the ISO/IEC 9899:1999
10103 standard provides for all three versions of math functions.

10104 The functions *ecvt()*, *fcvt()*, and *gcvt()* have been dropped from the ISO C standard since their
10105 capability is available through *sprintf()*. These are provided on XSI-conformant systems
10106 supporting the Legacy Option Group.

10107 FUTURE DIRECTIONS

10108 None.

10109 SEE ALSO

10110 The System Interfaces volume of IEEE Std. 1003.1-200x, *acos()*, *acosh()*, *asin()*, *atan()*, *atan2()*,
10111 *cbrt()*, *ceil()*, *cos()*, *cosh()*, *erf()*, *exp()*, *expm1()*, *fabs()*, *floor()*, *fmod()*, *frexp()*, *hypot()*, *ilogb()*,
10112 *isnan()*, *j0()*, *ldexp()*, *lgamma()*, *log()*, *log10()*, *log1p()*, *logb()*, *modf()*, *nextafter()*, *pow()*,
10113 *remainder()*, *rint()*, *scalb()*, *sin()*, *sinh()*, *sqrt()*, *tan()*, *tanh()*, *y0()*

10114 CHANGE HISTORY

10115 First released in Issue 1.

10116 Issue 4

10117 The constants M_E and MAXFLOAT are marked as extensions.

10118 The functions declared in this header are subdivided into those defined in the ISO C standard,
10119 and those defined only by The Open Group. Functions in the latter group are marked as
10120 extensions, as is the external variable *signgam*.

10121 The following changes are incorporated for alignment with the ISO C standard:

- 10122 • The description of HUGE_VAL is changed to indicate that this value is not necessarily
10123 representable as a **float**.
- 10124 • The function declarations in this header are expanded to full ISO C standard prototypes.

10125 Issue 4, Version 2

10126 The following change is incorporated for X/OPEN UNIX conformance:

- 10127 • The *acosh()*, *asinh()*, *atanh()*, *cbrt()*, *expm1()*, *ilogb()*, *log1p()*, *logb()*, *nextafter()*, *remainder()*,
10128 *rint()*, and *scalb()* functions are added to the list of functions declared in this header.

10129 Issue 6

10130 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

10131 **NAME**

10132 monetary.h — monetary types

10133 **SYNOPSIS**

10134 XSI #include <monetary.h>

10135

10136 **DESCRIPTION**10137 The <monetary.h> header shall define the following data types through **typedef**:10138 **size_t** As described in <stddef.h>.10139 **ssize_t** As described in <sys/types.h>.10140 The following shall be declared as a function and may also be defined as a macro. Function
10141 prototypes shall be provided for use with an ISO C standard compiler.10142 **ssize_t** strfmon(char *restrict, size_t, const char *restrict, ...);10143 **APPLICATION USAGE**

10144 None.

10145 **RATIONALE**

10146 None.

10147 **FUTURE DIRECTIONS**

10148 None.

10149 **SEE ALSO**10150 The System Interfaces volume of IEEE Std. 1003.1-200x, *strfmon()*10151 **CHANGE HISTORY**

10152 First released in Issue 4.

10153 **Issue 6**10154 The **restrict** keyword is added to the prototype for *strfmon()*.

10155 **NAME**10156 mqueue.h — message queues (**REALTIME**)10157 **SYNOPSIS**

10158 MSG #include <mqueue.h>

10159

10160 **DESCRIPTION**10161 The <mqueue.h> header shall define the **mqd_t** type, which is used for message queue
10162 descriptors. This is not an array type.10163 The <mqueue.h> header shall define the **sigevent** structure (as described in <signal.h>) and the
10164 **mq_attr** structure, which is used in getting and setting the attributes of a message queue.
10165 Attributes are initially set when the message queue is created. An **mq_attr** structure shall have at
10166 least the following fields:

10167	long	mq_flags	Message queue flags.
10168	long	mq_maxmsg	Maximum number of messages.
10169	long	mq_msgsize	Maximum message size.
10170	long	mq_curmsgs	Number of messages currently queued.

10171 The following shall be declared as functions and may also be declared as macros. Function
10172 prototypes shall be provided for use with an ISO C standard compiler.

```

10173 int      mq_close(mqd_t);
10174 int      mq_getattr(mqd_t, struct mq_attr *);
10175 int      mq_notify(mqd_t, const struct sigevent *);
10176 mqd_t    mq_open(const char *, int, ...);
10177 ssize_t  mq_receive(mqd_t, char *, size_t, unsigned *);
10178 int      mq_send(mqd_t, const char *, size_t, unsigned);
10179 int      mq_setattr(mqd_t, const struct mq_attr *restrict,
10180                  struct mq_attr *restrict);
10181 TMO int      mq_timedreceive(mqd_t, char *restrict, size_t,
10182                          unsigned *restrict, const struct timespec *restrict);
10183 int      mq_timedsend(mqd_t, const char *, size_t, unsigned,
10184                      const struct timespec *);
10185 int      mq_unlink(const char *);

```

10186 **Notes to Reviewers**10187 *This section with side shading will not appear in the final copy. - Ed.*10188 D3, XBD, ERN 163: The return type from mq_timedreceive() should be ssize_t and not int. An
10189 interpretation should be filed against .1d to bring this change into scope.10190 Inclusion of the <mqueue.h> header may make visible symbols defined in the headers <fcntl.h>,
10191 <signal.h>, <sys/types.h>, and <time.h>.

10192 **APPLICATION USAGE**

10193 None.

10194 **RATIONALE**

10195 None.

10196 **FUTURE DIRECTIONS**

10197 None.

10198 **SEE ALSO**

10199 <fcntl.h>, <signal.h>, <sys/types.h>, <time.h>, the System Interfaces volume of
10200 IEEE Std. 1003.1-200x, *mq_close()*, *mq_getattr()*, *mq_notify()*, *mq_open()*, *mq_receive()*, *mq_send()*,
10201 *mq_setattr()*, *mq_timedreceive()*, *mq_timedsend()*, *mq_unlink()*

10202 **CHANGE HISTORY**

10203 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

10204 **Issue 6**

10205 The <mqqueue.h> header is marked as part of the Message Passing option. |

10206 The *mq_timedreceive()* and *mq_timedsend()* functions are added for alignment with |
10207 IEEE Std. 1003.1d-1999. |

10208 The **restrict** keyword is added to the prototypes for *mq_setattr()* and *mq_timedreceive()*. |

10209 **NAME**

10210 ndbm.h — definitions for ndbm database operations

10211 **SYNOPSIS**10212 XSI `#include <ndbm.h>`

10213

10214 **DESCRIPTION**10215 The **<ndbm.h>** header shall define the **datum** type as a structure that includes at least the
10216 following members:10217 `void *dptr` A pointer to the application's data.10218 `size_t dsize` The size of the object pointed to by *dptr*.10219 The `size_t` type shall be defined through **typedef** as described in **<stddef.h>**.10220 The **<ndbm.h>** header shall define the **DBM** type through **typedef**.10221 The following constants shall be defined as possible values for the *store_mode* argument to
10222 *dbm_store()*:10223 **DBM_INSERT** Insertion of new entries only.10224 **DBM_REPLACE** Allow replacing existing entries.10225 The following shall be declared as functions and may also be defined as macros. Function
10226 prototypes shall be provided for use with an ISO C standard compiler.10227 `int dbm_clearerr(DBM *);`10228 `void dbm_close(DBM *);`10229 `int dbm_delete(DBM *, datum);`10230 `int dbm_error(DBM *);`10231 `datum dbm_fetch(DBM *, datum);`10232 `datum dbm_firstkey(DBM *);`10233 `datum dbm_nextkey(DBM *);`10234 `DBM *dbm_open(const char *, int, mode_t);`10235 `int dbm_store(DBM *, datum, datum, int);`10236 The `mode_t` type shall be defined through **typedef** as described in **<sys/types.h>**.10237 **APPLICATION USAGE**

10238 None.

10239 **RATIONALE**

10240 None.

10241 **FUTURE DIRECTIONS**

10242 None.

10243 **SEE ALSO**10244 The System Interfaces volume of IEEE Std. 1003.1-200x, *dbm_clearerr()*10245 **CHANGE HISTORY**

10246 First released in Issue 4, Version 2.

10247 **Issue 5**10248 References to the definitions of `size_t` and `mode_t` are added to the DESCRIPTION.

10249 **NAME**

10250 net/if.h — sockets local interfaces

10251 **SYNOPSIS**

10252 #include <net/if.h>

10253 **DESCRIPTION**10254 The <net/if.h> header shall define the **if_nameindex** structure that includes at least the
10255 following members:10256 unsigned if_index Numeric index of the interface.
10257 char *if_name Null-terminated name of the interface.10258 The <net/if.h> header shall define the following macro for the length of a buffer containing an
10259 interface name (including the terminating NULL character):10260 **IF_NAMESIZE** Interface name length.10261 The following shall be declared as functions, and may also be defined as macros. Function
10262 prototypes shall be provided for use with an ISO C standard compiler.10263 unsigned if_nametoindex(const char*);
10264 char *if_indextoname(unsigned, char*);
10265 struct if_nameindex *if_nameindex(void);
10266 void if_freenameindex(struct if_nameindex*);10267 **APPLICATION USAGE**

10268 None.

10269 **RATIONALE**

10270 None.

10271 **FUTURE DIRECTIONS**

10272 None.

10273 **SEE ALSO**10274 The System Interfaces volume of IEEE Std. 1003.1-200x, *if_freenameindex()*, *if_indextoname()*,
10275 *if_nameindex()*, *if_nametoindex()*10276 **CHANGE HISTORY**

10277 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10278 **NAME**

10279 netdb.h — definitions for network database operations

10280 **SYNOPSIS**

10281 #include <netdb.h>

10282 **DESCRIPTION**10283 The <netdb.h> header may make available the **in_port_t** type and the **in_addr_t** type as defined
10284 in <netinet/in.h>.10285 The <netdb.h> header shall define the **hostent** structure that includes at least the following
10286 members:

10287	char	*h_name	Official name of the host.
10288	char	**h_aliases	A pointer to an array of pointers to
10289			alternative host names, terminated by a
10290			null pointer.
10291	int	h_addrtype	Address type.
10292	int	h_length	The length, in bytes, of the address.
10293	char	**h_addr_list	A pointer to an array of pointers to network
10294			addresses (in network byte order) for the host,
10295			terminated by a null pointer.

10296 The <netdb.h> header shall define the **netent** structure that includes at least the following
10297 members:

10298	char	*n_name	Official, fully-qualified (including the
10299			domain) name of the host.
10300	char	**n_aliases	A pointer to an array of pointers to
10301			alternative network names, terminated by a
10302			null pointer.
10303	int	n_addrtype	The address type of the network.
10304	uint32_t	n_net	The network number, in host byte order.

10305 The **uint32_t** type shall be defined as described in <inttypes.h>.10306 The <netdb.h> header shall define the **protoent** structure that includes at least the following
10307 members:

10308	char	*p_name	Official name of the protocol.
10309	char	**p_aliases	A pointer to an array of pointers to
10310			alternative protocol names, terminated by
10311			a null pointer.
10312	int	p_proto	The protocol number.

10313 The <netdb.h> header shall define the **servent** structure that includes at least the following
10314 members:

10315	char	*s_name	Official name of the service.
10316	char	**s_aliases	A pointer to an array of pointers to
10317			alternative service names, terminated by
10318			a null pointer.
10319	int	s_port	The port number at which the service
10320			resides, in network byte order.
10321	char	*s_proto	The name of the protocol to use when
10322			contacting the service.

10323 The <netdb.h> header shall define the IPPORT_RESERVED macro with the value of the highest
 10324 reserved Internet port number.

10325 When the <netdb.h> header is included, *h_errno* shall be available as a modifiable *l*-value of type
 10326 **int**. It is unspecified whether *h_errno* is a macro or an identifier declared with external linkage.

10327 The <netdb.h> header shall define the following macros for use as error values for
 10328 *gethostbyaddr()*, *gethostbyname()*, *getipnodebyaddr()*, and *getipnodebyname()*:

- 10329 HOST_NOT_FOUND
- 10330 NO_DATA
- 10331 NO_RECOVERY
- 10332 TRY_AGAIN

10333 The <netdb.h> header shall define the following macros that evaluate to bitwise-distinct integer
 10334 constants, for use in the *flags* argument of *getipnodebyname()*:

10335 IP6 AI_V4MAPPED IPv4-mapped IPv6 addresses are acceptable.

10336 AI_ALL Return all addresses: IPv6 and IPv4-mapped IPv6.

10337 AI_ADDRCONFIG
 10338 Return addresses depending on what source addresses are configured.

10339 The <netdb.h> header shall define the AI_DEFAULT macro, which evaluates to the logical OR of
 10340 AI_V4MAPPED and AI_ADDRCONFIG.

10341 **Address Information Structure**

10342 The <netdb.h> header shall define the **addrinfo** structure that includes at least the following
 10343 members:

10344	int	ai_flags	Input flags.
10345	int	ai_family	Address family of socket.
10346	int	ai_socktype	Socket type.
10347	int	ai_protocol	Protocol of socket.
10348	socklen_t	ai_addrlen	Length of socket address.
10349	struct sockaddr	*ai_addr	Socket address of socket.
10350	char	*ai_canonname	Canonical name of service location.
10351	struct addrinfo	*ai_next	Pointer to next in list.

10352 The <netdb.h> header shall define the following macros that evaluate to bitwise-distinct integer
 10353 constants for use in the *flags* field of the **addrinfo** structure:

10354 AI_PASSIVE Socket address is intended for *bind()*.

10355 AI_CANONNAME
 10356 Request for canonical name.

10357 AI_NUMERICHOST
 10358 Return numeric host address as name.

10359 The <netdb.h> header shall define the following macros that evaluate to bitwise-distinct integer
 10360 constants for use in the *flags* argument to *getnameinfo()*:

10361 NI_NOFQDN Only the nodename portion of the FQDN is returned for local hosts.

10362 NI_NUMERICHOST
 10363 The numeric form of the node's address is returned instead of its name.

10364 NI_NAMEREQD Return an error if the node's name cannot be located in the database.

10365 NI_NUMERICSERV
10366 The numeric form of the service address is returned instead of its name.

10367 NI_DGRAM Indicates that the service is a datagram service (SOCK_DGRAM).

10368 **Address Information Errors**

10369 The <netdb.h> header shall define the following macros for use as error values for *getaddrinfo()*
10370 and *getnameinfo()*:

10371 EAI_AGAIN The name could not be resolved at this time. Future attempts may succeed.

10372 EAI_BADFLAGS The flags had an invalid value.

10373 EAI_FAIL A non-recoverable error occurred.

10374 EAI_FAMILY The address family was not recognized or the address length was invalid for
10375 the specified family.

10376 EAI_MEMORY There was a memory allocation failure.

10377 EAI_NONAME The name does not resolve for the supplied parameters.

10378 NI_NAMEREQD is set and the host's name cannot be located, or both
10379 *nodename* and *servname* were null.

10380 EAI_SERVICE The service passed was not recognized for the specified socket type.

10381 EAI_SOCKTYPE The intended socket type was not recognized.

10382 EAI_SYSTEM A system error occurred. The error code can be found in *errno*.

10383 The following shall be declared as functions, and may also be defined as macros. Function
10384 prototypes shall be provided for use with an ISO C standard compiler.

```

10385 void          endhostent(void);
10386 void          endnetent(void);
10387 void          endprotoent(void);
10388 void          endservent(void);
10389 void          freeaddrinfo(struct addrinfo *);
10390 void          freehostent(struct hostent *);
10391 char          *gai_strerror(int);
10392 int           getaddrinfo(const char *, const char *,
10393                          const struct addrinfo *, struct addrinfo **);
10394 struct hostent *gethostbyaddr(const void *, socklen_t, int);
10395 struct hostent *gethostbyname(const char *);
10396 struct hostent *gethostent(void);
10397 struct hostent *getipnodebyaddr(const void *restrict, socklen_t, int,
10398                                int *restrict);
10399 struct hostent *getipnodebyname(const char *, int, int, int *);
10400 int           getnameinfo(const struct sockaddr *, socklen_t,
10401                          char *, socklen_t, char *, socklen_t, unsigned);
10402 struct netent *getnetbyaddr(uint32_t, int);
10403 struct netent *getnetbyname(const char *);
10404 struct netent *getnetent(void);
10405 struct protoent *getprotobyname(const char *);
10406 struct protoent *getprotobynumber(int);
10407 struct protoent *getprotoent(void);

```



```
10408     struct servent     *getservbyname(const char *, const char *);
10409     struct servent     *getservbyport(int, const char *);
10410     struct servent     *getservent(void);
10411     void                sethostent(int);
10412     void                setnetent(int);
10413     void                setprotoent(int);
10414     void                setservent(int);

10415     The type socklen_t shall be defined through typedef as described in <sys/socket.h>.

10416     Inclusion of the <netdb.h> header may also make visible all symbols from <netinet/in.h> and
10417     <inttypes.h>.

10418 APPLICATION USAGE
10419     None.

10420 RATIONALE
10421     None.

10422 FUTURE DIRECTIONS
10423     None.

10424 SEE ALSO
10425     <netinet/in.h>, <inttypes.h>, <sys/socket.h>, the System Interfaces volume of
10426     IEEE Std. 1003.1-200x, bind(), endhostent(), endnetent(), endprotoent(), endservent(), getaddrinfo(),
10427     getnameinfo()

10428 CHANGE HISTORY
10429     First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10430     The restrict keyword is added to the prototype for getipnodebyaddr().
```

10431 NAME

10432 netinet/in.h — Internet protocol family

10433 SYNOPSIS

10434 #include <netinet/in.h>

10435 DESCRIPTION

10436 The <netinet/in.h> header shall define the following types through **typedef**:

10437 **in_port_t** An unsigned integer type of exactly 16 bits.

10438 **in_addr_t** An unsigned integer type of exactly 32 bits.

10439 The **sa_family_t** type shall be defined as described in <sys/socket.h>.

10440 The **uint32_t** type shall be defined as described in <inttypes.h>. Inclusion of the <netinet/in.h>
10441 header may also make visible all symbols from <inttypes.h>.

10442 The <netinet/in.h> header shall define the **in_addr** structure that includes at least the following
10443 member:

10444 in_addr_t s_addr

10445 The <netinet/in.h> header shall define the **sockaddr_in** structure that includes at least the
10446 following members:

10447 sa_family_t sin_family
10448 in_port_t sin_port
10449 struct in_addr sin_addr
10450 unsigned char sin_zero[8]

10451 The **sockaddr_in** structure is used to store addresses for the Internet protocol family. Values of
10452 this type shall be cast by applications to **struct sockaddr** for use with socket functions.

10453 IP6 The <netinet/in.h> header shall define the **in6_addr** structure that contains at least the following
10454 member:

10455 uint8_t s6_addr[16]

10456 This array is used to contain a 128-bit IPv6 address, stored in network byte order.

10457 The <netinet/in.h> header shall define the **sockaddr_in6** structure that includes at least the
10458 following members:

10459 sa_family_t sin6_family AF_INET6.
10460 in_port_t sin6_port Port number.
10461 uint32_t sin6_flowinfo IPv6 traffic class and flow information.
10462 struct in6_addr sin6_addr IPv6 address.
10463 uint32_t sin6_scope_id Set of interfaces for a scope.

10464 The **sockaddr_in6** structure shall be set to zero by an application prior to using it, since
10465 implementations are free to have additional, implementation-defined fields in **sockaddr_in6**.

10466 The *sin6_scope_id* field is a 32-bit integer that identifies a set of interfaces as appropriate for the
10467 scope of the address carried in the *sin6_addr* field. For a link scope *sin6_addr*, *sin6_scope_id* would
10468 be an interface index. For a site scope *sin6_addr*, *sin6_scope_id* would be a site identifier. The
10469 mapping of *sin6_scope_id* to an interface or set of interfaces is implementation-defined.

10470 The <netinet/in.h> header shall declare the following external variable:

10471 struct in6_addr in6addr_any

10472 This variable is initialized by the system to contain the wildcard IPv6 address. The
 10473 <netinet/in.h> header also defines the IN6ADDR_ANY_INIT macro. This macro must be
 10474 constant at compile time and can be used to initialize a variable of type **struct in6_addr** to the
 10475 IPv6 wildcard address.

10476 The <netinet/in.h> header shall declare the following external variable:

```
10477 struct in6_addr in6addr_loopback
```

10478 This variable is initialized by the system to contain the loopback IPv6 address. The
 10479 <netinet/in.h> header also defines the IN6ADDR_LOOPBACK_INIT macro. This macro must be
 10480 constant at compile time and can be used to initialize a variable of type **struct in6_addr** to the
 10481 IPv6 loopback address.

10482 The <netinet/in.h> header shall define the **ipv6_mreq** structure that includes at least the
 10483 following members:

```
10484 struct in6_addr  ipv6mr_multiaddr  IPv6 multicast address.  

10485 unsigned        ipv6mr_interface  Interface index.
```

10486

10487 The <netinet/in.h> header shall define the following macros for use as values of the *level*
 10488 argument of *getsockopt()* and *setsockopt()*:

10489 IPPROTO_IP Internet protocol.

10490 ^{IP6} IPPROTO_IPV6 Internet Protocol Version 6.

10491 IPPROTO_ICMP Control message protocol.

10492 IPPROTO_TCP Transmission control protocol.

10493 IPPROTO_UDP User datagram protocol.

10494 The <netinet/in.h> header shall define the following macros for use as destination addresses for
 10495 *connect()*, *sendmsg()*, and *sendto()*:

10496 INADDR_ANY IPv4 local host address.

10497 INADDR_BROADCAST IPv4 broadcast address.

10498 The <netinet/in.h> header shall define the following macro to help applications declare buffers
 10499 of the proper size to store IPv4 addresses in string form:

10500 INET_ADDRSTRLEN 16.

10501 The *htonl()*, *htons()*, *ntohl()*, and *ntohs()* functions shall be available as defined in <arpa/inet.h>.
 10502 Inclusion of the <netinet/in.h> header may also make visible all symbols from <arpa/inet.h>.

10503 ^{IP6} The <netinet/in.h> header shall define the following macro to help applications declare buffers
 10504 of the proper size to store IPv6 addresses in string form:

10505 INET6_ADDRSTRLEN 46.

10506 The <netinet/in.h> header shall define the following macros, with distinct integral values, for
 10507 use in the *option_name* argument in the *getsockopt()* or *setsockopt()* functions at protocol level
 10508 IPPROTO_IPV6:

10509 IPV6_JOIN_GROUP Join a multicast group.

10510 IPV6_LEAVE_GROUP Quit a multicast group.

10511 IPV6_MULTICAST_HOPS
10512 Multicast hop limit.

10513 IPV6_MULTICAST_IF Interface to use for outgoing multicast packets.

10514 IPV6_MULTICAST_LOOP
10515 Multicast packets are delivered back to the local application.

10516 IPV6_UNICAST_HOPS Unicast hop limit.

10517 The <netinet/in.h> header shall define the following macros that test for special IPv6 addresses.
10518 Each macro is of type **int** and takes a single argument of type **const struct in6_addr***:

10519 IN6_IS_ADDR_UNSPECIFIED
10520 Unspecified address.

10521 IN6_IS_ADDR_LOOPBACK
10522 Loopback address.

10523 IN6_IS_ADDR_MULTICAST
10524 Multicast address.

10525 IN6_IS_ADDR_LINKLOCAL
10526 Unicast link-local address.

10527 IN6_IS_ADDR_SITELOCAL
10528 Unicast site-local address.

10529 IN6_IS_ADDR_V4MAPPED
10530 IPv4 mapped address.

10531 IN6_IS_ADDR_V4COMPAT
10532 IPv4-compatible address.

10533 IN6_IS_ADDR_MC_NODELOCAL
10534 Multicast node-local address.

10535 IN6_IS_ADDR_MC_LINKLOCAL
10536 Multicast link-local address.

10537 IN6_IS_ADDR_MC_SITELOCAL
10538 Multicast site-local address.

10539 IN6_IS_ADDR_MC_ORGLOCAL
10540 Multicast organization-local address.

10541 IN6_IS_ADDR_MC_GLOBAL
10542 Multicast global address.

10543 IN6_IS_ADDR_LINKLOCAL and IN6_IS_ADDR_SITELOCAL return true only for the two
10544 local-use IPv6 unicast addresses. They do not return true for multicast addresses of either link-
10545 local or site-local scope.

10546 **APPLICATION USAGE**

10547 None.

10548 **RATIONALE**

10549 None.

10550 **FUTURE DIRECTIONS**

10551 None.

10552 **SEE ALSO**

10553 <arpa/inet.h>, <inttypes.h>, <sys/socket.h>, the System Interfaces volume of |
10554 IEEE Std. 1003.1-200x, *connect()*, *getsockopt()*, *htonl()*, *htons()*, *ntohl()*, *ntohs()*, *sendmsg()*,
10555 *sendto()*, *setsockopt()*

10556 **CHANGE HISTORY**

10557 First released in Issue 6. Derived from the XNS, Issue 5.2 specification. |

10558 **NAME**

10559 netinet/tcp.h — definitions for the Internet Transmission Control Protocol (TCP)

10560 **SYNOPSIS**

10561 #include <netinet/tcp.h>

10562 **DESCRIPTION**

10563 The **<netinet/tcp.h>** header shall define the following macro for use as a socket option at the
10564 IPPROTO_TCP level:

10565 TCP_NODELAY Avoid coalescing of small segments.

10566 The macro shall be defined in the header. The implementation need not allow the value of the
10567 option to be set via *setsockopt()* or retrieved via *getsockopt()*.

10568 **APPLICATION USAGE**

10569 None.

10570 **RATIONALE**

10571 None.

10572 **FUTURE DIRECTIONS**

10573 None.

10574 **SEE ALSO**

10575 **<sys/socket.h>**, the System Interfaces volume of IEEE Std. 1003.1-200x, *getsockopt()*, *setsockopt()*

10576 **CHANGE HISTORY**

10577 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10578 **NAME**

10579 nl_types.h — data types

10580 **SYNOPSIS**

10581 XSI #include <nl_types.h>

10582

10583 **DESCRIPTION**

10584 The <nl_types.h> header shall contain definitions of at least the following types:

10585 **nl_catd** Used by the message catalog functions *catopen()*, *catgets()*, and *catclose()*
 10586 to identify a catalog descriptor.

10587 **nl_item** Used by *nl_langinfo()* to identify items of *langinfo* data. Values of objects
 10588 of type **nl_item** are defined in <langinfo.h>.

10589 The <nl_types.h> header shall contain definitions of at least the following constants:

10590 **NL_SETD** Used by *genocat* when no *\$set* directive is specified in a message text source
 10591 file; see the Internationalization Guide. This constant can be passed as the
 10592 value of *set_id* on subsequent calls to *catgets()* (that is, to retrieve
 10593 messages from the default message set). The value of **NL_SETD** is
 10594 implementation-defined.

10595 **NL_CAT_LOCALE** Value that must be passed as the *offlag* argument to *catopen()* to ensure
 10596 that message catalog selection depends on the *LC_MESSAGES* locale
 10597 category, rather than directly on the *LANG* environment variable.

10598 The following shall be declared as functions and may also be defined as macros. Function
 10599 prototypes shall be provided for use with an ISO C standard compiler.

```
10600 int      catclose(nl_catd);
10601 char    *catgets(nl_catd, int, int, const char *);
10602 nl_catd catopen(const char *, int);
```

10603 **APPLICATION USAGE**

10604 None.

10605 **RATIONALE**

10606 None.

10607 **FUTURE DIRECTIONS**

10608 None.

10609 **SEE ALSO**

10610 <langinfo.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *catclose()*, *catgets()*,
 10611 *catopen()*, *nl_langinfo()*, the Shell and Utilities volume of IEEE Std. 1003.1-200x, *genocat*

10612 **CHANGE HISTORY**

10613 First released in Issue 2.

10614 **Issue 4**

10615 The following change is incorporated for alignment with the ISO C standard:

- 10616 • The function declarations in this header are expanded to full ISO C standard prototypes.

10617 **NAME**

10618 poll.h — definitions for the poll() function

10619 **SYNOPSIS**

10620 XSI #include <poll.h>

10621

10622 **DESCRIPTION**10623 The <poll.h> header shall define the **pollfd** structure that includes at least the following
10624 members:

10625 int fd The following descriptor being polled.

10626 short events The input event flags (see below).

10627 short revents The output event flags (see below).

10628 The <poll.h> header shall define the following type through **typedef**:10629 **nfds_t** An unsigned integer type used for the number of file descriptors.10630 The following symbolic constants shall be defined, zero or more of which may be OR'ed together
10631 to form the *events* or *revents* members in the **pollfd** structure:

10632 POLLIN Same effect as POLLRDNORM | POLLRDBAND.

10633 POLLRDNORM Data on priority band 0 may be read.

10634 POLLRDBAND Data on priority bands greater than 0 may be read.

10635 POLLPRI High priority data may be read.

10636 POLLOUT Same value as POLLWRNORM.

10637 POLLWRNORM Data on priority band 0 may be written.

10638 POLLWRBAND Data on priority bands greater than 0 may be written. This event only
10639 examines bands that have been written to at least once.10640 POLLERR An error has occurred (*revents* only).10641 POLLHUP Device has been disconnected (*revents* only).10642 POLLNVAL Invalid *fd* member (*revents* only).10643 The <poll.h> header shall declare the following function which may also be defined as a macro.
10644 Function prototypes shall be provided for use with an ISO C standard compiler.

10645 int poll(struct pollfd[], nfds_t, int);

10646 **APPLICATION USAGE**

10647 None.

10648 **RATIONALE**

10649 None.

10650 **FUTURE DIRECTIONS**

10651 None.

10652 **SEE ALSO**10653 The System Interfaces volume of IEEE Std. 1003.1-200x, *poll()*

10654 **CHANGE HISTORY**

10655 First released in Issue 4, Version 2.

10656 NAME

10657 pthread.h — threads

10658 SYNOPSIS

10659 THR #include <pthread.h>

10660

10661 DESCRIPTION

10662 The <pthread.h> header shall define the following symbols:

- 10663 BAR PTHREAD_BARRIER_SERIAL_THREAD
- 10664 PTHREAD_CANCEL_ASYNCHRONOUS
- 10665 PTHREAD_CANCEL_ENABLE
- 10666 PTHREAD_CANCEL_DEFERRED
- 10667 PTHREAD_CANCEL_DISABLE
- 10668 PTHREAD_CANCELED
- 10669 PTHREAD_COND_INITIALIZER
- 10670 PTHREAD_CREATE_DETACHED
- 10671 PTHREAD_CREATE_JOINABLE
- 10672 PTHREAD_EXPLICIT_SCHED
- 10673 PTHREAD_INHERIT_SCHED
- 10674 XSI PTHREAD_MUTEX_DEFAULT
- 10675 PTHREAD_MUTEX_ERRORCHECK
- 10676 PTHREAD_MUTEX_INITIALIZER
- 10677 XSI PTHREAD_MUTEX_NORMAL
- 10678 PTHREAD_MUTEX_RECURSIVE
- 10679 PTHREAD_ONCE_INIT
- 10680 TPP|TPI PTHREAD_PRIO_INHERIT
- 10681 PTHREAD_PRIO_NONE
- 10682 PTHREAD_PRIO_PROTECT
- 10683 PTHREAD_PROCESS_SHARED
- 10684 PTHREAD_PROCESS_PRIVATE
- 10685 TPS PTHREAD_SCOPE_PROCESS
- 10686 PTHREAD_SCOPE_SYSTEM

10687

10688 The following types shall be defined as described in <sys/types.h>:

- 10689 pthread_attr_t
- 10690 BAR pthread_barrier_t
- 10691 pthread_barrierattr_t
- 10692 pthread_cond_t
- 10693 pthread_condattr_t
- 10694 pthread_key_t
- 10695 pthread_mutex_t
- 10696 pthread_mutexattr_t
- 10697 pthread_once_t
- 10698 pthread_rwlock_t
- 10699 pthread_rwlockattr_t
- 10700 SPI pthread_spinlock_t
- 10701 pthread_t

10702 The following shall be declared as functions and may also be declared as macros. Function
10703 prototypes shall be provided for use with an ISO C standard compiler.

```

10704     int   pthread_atfork(void (*)(void), void (*)(void),
10705                          void (*)(void));
10706     int   pthread_attr_destroy(pthread_attr_t *);
10707     int   pthread_attr_getdetachstate(const pthread_attr_t *, int *);
10708 XSI    int   pthread_attr_getguardsize(const pthread_attr_t *restrict,
10709                                       size_t *restrict);
10710 TPS    int   pthread_attr_getinheritsched(const pthread_attr_t *restrict,
10711                                           int *restrict);
10712     int   pthread_attr_getschedparam(const pthread_attr_t *restrict,
10713                                     struct sched_param *restrict);
10714 TPS    int   pthread_attr_getschedpolicy(const pthread_attr_t *restrict,
10715                                          int *restrict);
10716 TPS    int   pthread_attr_getscope(const pthread_attr_t *restrict,
10717                                   int *restrict);
10718 TSA    int   pthread_attr_getstackaddr(const pthread_attr_t *restrict,
10719                                       void **restrict);
10720     int   pthread_attr_getstacksize(const pthread_attr_t *restrict,
10721                                   size_t *restrict);
10722     int   pthread_attr_init(pthread_attr_t *);
10723     int   pthread_attr_setdetachstate(pthread_attr_t *, int);
10724 XSI    int   pthread_attr_setguardsize(pthread_attr_t *, size_t);
10725 TPS    int   pthread_attr_setinheritsched(pthread_attr_t *, int);
10726     int   pthread_attr_setschedparam(pthread_attr_t *restrict,
10727                                     const struct sched_param *restrict);
10728 TPS    int   pthread_attr_setschedpolicy(pthread_attr_t *, int);
10729     int   pthread_attr_setscope(pthread_attr_t *, int);
10730 TSA    int   pthread_attr_setstackaddr(pthread_attr_t *, void *);
10731     int   pthread_attr_setstacksize(pthread_attr_t *, size_t);
10732 BAR    int   pthread_barrier_destroy(pthread_barrier_t *);
10733     int   pthread_barrier_init(pthread_barrier_t *restrict,
10734                               const pthread_barrierattr_t *restrict, unsigned);
10735     int   pthread_barrier_wait(pthread_barrier_t *);
10736     int   pthread_barrierattr_destroy(pthread_barrierattr_t *);
10737     int   pthread_barrierattr_getpshared(const pthread_barrierattr_t *restrict,
10738                                         int *restrict);
10739     int   pthread_barrierattr_init(pthread_barrierattr_t *);
10740     int   pthread_barrierattr_setpshared(pthread_barrierattr_t *, int);
10741     int   pthread_cancel(pthread_t);
10742     void  pthread_cleanup_push(void (*)(void *), void *);
10743     void  pthread_cleanup_pop(int);
10744     int   pthread_cond_broadcast(pthread_cond_t *);
10745     int   pthread_cond_destroy(pthread_cond_t *);
10746     int   pthread_cond_init(pthread_cond_t *restrict,
10747                             const pthread_condattr_t *restrict);
10748     int   pthread_cond_signal(pthread_cond_t *);
10749     int   pthread_cond_timedwait(pthread_cond_t *restrict,
10750                                 pthread_mutex_t *restrict, const struct timespec *restrict);
10751     int   pthread_cond_wait(pthread_cond_t *restrict,
10752                             pthread_mutex_t *restrict);
10753     int   pthread_condattr_destroy(pthread_condattr_t *);
10754 CS     int   pthread_condattr_getclock(const pthread_condattr_t *restrict,
10755                                       clockid_t *restrict);

```

```

10756     int    pthread_condattr_getpshared(const pthread_condattr_t *restrict,
10757                                     int *restrict);
10758     int    pthread_condattr_init(pthread_condattr_t *);
10759 CS     int    pthread_condattr_setclock(pthread_condattr_t *, clockid_t);
10760     int    pthread_condattr_setpshared(pthread_condattr_t *, int);
10761     int    pthread_create(pthread_t *restrict, const pthread_attr_t *restrict,
10762                          void *(*)(void *), void *);
10763     int    pthread_detach(pthread_t);
10764     int    pthread_equal(pthread_t, pthread_t);
10765     void   pthread_exit(void *);
10766 XSI   int    pthread_getconcurrency(void);
10767 TCT   int    pthread_getcpuclockid(pthread_t, clockid_t *);
10768 TPS   int    pthread_getschedparam(pthread_t, int *restrict,
10769                                   struct sched_param *restrict);
10770     void *pthread_getspecific(pthread_key_t);
10771     int    pthread_join(pthread_t, void **);
10772     int    pthread_key_create(pthread_key_t *, void (*)(void *));
10773     int    pthread_key_delete(pthread_key_t);
10774     int    pthread_kill(pthread_t, int);
10775     int    pthread_mutex_destroy(pthread_mutex_t *);
10776 TPP   int    pthread_mutex_getprioceiling(const pthread_mutex_t *restrict,
10777                                         int *restrict);
10778     int    pthread_mutex_init(pthread_mutex_t *restrict,
10779                              const pthread_mutexattr_t *restrict);
10780     int    pthread_mutex_lock(pthread_mutex_t *);
10781 TPP   int    pthread_mutex_setprioceiling(pthread_mutex_t *restrict, int,
10782                                         int *restrict);
10783 TMO   int    pthread_mutex_timedlock(pthread_mutex_t *,
10784                                     const struct timespec *);
10785     int    pthread_mutex_trylock(pthread_mutex_t *);
10786     int    pthread_mutex_unlock(pthread_mutex_t *);
10787     int    pthread_mutexattr_destroy(pthread_mutexattr_t *);
10788 TPP|TPI int    pthread_mutexattr_getprioceiling(const pthread_mutexattr_t *restrict,
10789                                               int *restrict);
10790     int    pthread_mutexattr_getprotocol(const pthread_mutexattr_t *restrict,
10791                                         int *restrict);
10792     int    pthread_mutexattr_getpshared(const pthread_mutexattr_t *restrict,
10793                                         int *restrict);
10794 XSI   int    pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict,
10795                                       int *restrict);
10796     int    pthread_mutexattr_init(pthread_mutexattr_t *);
10797 TPP|TPI int    pthread_mutexattr_setprioceiling(pthread_mutexattr_t *, int);
10798     int    pthread_mutexattr_setprotocol(pthread_mutexattr_t *, int);
10799     int    pthread_mutexattr_setpshared(pthread_mutexattr_t *, int);
10800 XSI   int    pthread_mutexattr_settype(pthread_mutexattr_t *, int);
10801     int    pthread_once(pthread_once_t *, void (*)(void));
10802     int    pthread_rwlock_destroy(pthread_rwlock_t *);
10803     int    pthread_rwlock_init(pthread_rwlock_t *restrict,
10804                               const pthread_rwlockattr_t *restrict);
10805     int    pthread_rwlock_rdlock(pthread_rwlock_t *);
10806     int    pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict,
10807                                       const struct timespec *restrict);

```

```

10808     int    pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict,
10809         const struct timespec *restrict);
10810     int    pthread_rwlock_tryrdlock(pthread_rwlock_t *);
10811     int    pthread_rwlock_trywrlock(pthread_rwlock_t *);
10812     int    pthread_rwlock_unlock(pthread_rwlock_t *);
10813     int    pthread_rwlock_wrlock(pthread_rwlock_t *);
10814     int    pthread_rwlockattr_destroy(pthread_rwlockattr_t *);
10815     int    pthread_rwlockattr_getpshared(const pthread_rwlockattr_t *restrict,
10816         int *restrict);
10817     int    pthread_rwlockattr_init(pthread_rwlockattr_t *);
10818     int    pthread_rwlockattr_setpshared(pthread_rwlockattr_t *, int);
10819     pthread_t
10820         pthread_self(void);
10821     int    pthread_setcancelstate(int, int *);
10822     int    pthread_setcanceltype(int, int *);
10823 XSI    int    pthread_setconcurrency(int);
10824 TPS    int    pthread_setschedparam(pthread_t, int,
10825         const struct sched_param *);
10826     int    pthread_setspecific(pthread_key_t, const void *);
10827     int    pthread_sigmask(int, const sigset_t *restrict, sigset_t *restrict);
10828 SPI    int    pthread_spin_destroy(pthread_spinlock_t *);
10829     int    pthread_spin_init(pthread_spinlock_t *, int);
10830     int    pthread_spin_lock(pthread_spinlock_t *);
10831     int    pthread_spin_trylock(pthread_spinlock_t *);
10832     int    pthread_spin_unlock(pthread_spinlock_t *);
10833     void   pthread_testcancel(void);

```

10834 XSI **Inclusion of the <pthread.h> header shall make symbols defined in the headers <sched.h> and**
 10835 **<time.h> visible.**

10836 **APPLICATION USAGE**

10837 An interpretation request has been filed with IEEE PASC concerning requirements for visibility
 10838 of symbols in this header.

10839 **RATIONALE**

10840 None.

10841 **FUTURE DIRECTIONS**

10842 None.

10843 **SEE ALSO**

10844 <sched.h>, <time.h>, the System Interfaces volume of IEEE Std. 1003.1-200x,
 10845 *pthread_attr_getguardsize()*, *pthread_attr_init()*, *pthread_attr_setscope()*, *pthread_barrier_destroy()*,
 10846 *pthread_barrier_init()*, *pthread_barrier_wait()*, *pthread_barrierattr_destroy()*,
 10847 *pthread_barrierattr_getpshared()*, *pthread_barrierattr_init()*, *pthread_barrierattr_setpshared()*,
 10848 *pthread_cancel()*, *pthread_cleanup_pop()*, *pthread_cond_init()*, *pthread_cond_signal()*,
 10849 *pthread_cond_wait()*, *pthread_condattr_getclock()*, *pthread_condattr_init()*,
 10850 *pthread_condattr_setclock()*, *pthread_create()*, *pthread_detach()*, *pthread_equal()*, *pthread_exit()*,
 10851 *pthread_getconcurrency()*, *pthread_getcpuclockid()*, *pthread_getschedparam()*, *pthread_join()*,
 10852 *pthread_key_create()*, *pthread_key_delete()*, *pthread_mutex_init()*, *pthread_mutex_lock()*,
 10853 *pthread_mutex_setprioceiling()*, *pthread_mutex_timedlock()*, *pthread_mutexattr_init()*,
 10854 *pthread_mutexattr_gettype()*, *pthread_mutexattr_setprotocol()*, *pthread_once()*,
 10855 *pthread_rwlock_destroy()*, *pthread_rwlock_init()*, *pthread_rwlock_rdlock()*,
 10856 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_rwlock_tryrdlock()*,
 10857 *pthread_rwlock_trywrlock()*, *pthread_rwlock_unlock()*, *pthread_rwlock_wrlock()*,

10858 *pthread_rwlockattr_destroy()*, *pthread_rwlockattr_getpshared()*, *pthread_rwlockattr_init()*,
 10859 *pthread_rwlockattr_setpshared()*, *pthread_self()*, *pthread_setcancelstate()*, *pthread_setspecific()*,
 10860 *pthread_spin_destroy()*, *pthread_spin_init()*, *pthread_spin_lock()*, *pthread_spin_trylock()*,
 10861 *pthread_spin_unlock()*

10862 **CHANGE HISTORY**

10863 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

10864 **Issue 6**

10865 The RTT margin markers are now broken out into their POSIX options.

10866 The Open Group corrigenda item U021/9 has been applied, correcting the prototype for the
 10867 *pthread_cond_wait()* function.

10868 The Open Group corrigenda item U026/2 has been applied correcting the prototype for the
 10869 *pthread_setschedparam()* function so that its second argument is of type **int**.

10870 The *pthread_getcpuclockid()* and *pthread_mutex_timedlock()* functions are added for alignment
 10871 with IEEE Std. 1003.1d-1999.

10872 The following functions are added for alignment with IEEE Std. 1003.1j-2000:

10873 *pthread_barrier_destroy()*, *pthread_barrier_init()*, *pthread_barrier_wait()*,
 10874 *pthread_barrierattr_destroy()*, *pthread_barrierattr_getpshared()*, *pthread_barrierattr_init()*,
 10875 *pthread_barrierattr_setpshared()*, *pthread_condattr_getclock()*, *pthread_condattr_setclock()*,
 10876 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_spin_destroy()*,
 10877 *pthread_spin_init()*, *pthread_spin_lock()*, *pthread_spin_trylock()*, and *pthread_spin_unlock()*.

10878 PTHREAD_RWLOCK_INITIALIZER is deleted for alignment with IEEE Std. 1003.1j-2000.

10879 Functions previously marked as part of the Read-Write Locks option are now moved to the
 10880 Threads option.

10881 The **restrict** keyword is added to the prototypes for *pthread_attr_getguardsize()*,
 10882 *pthread_attr_getinheritsched()*, *pthread_attr_getschedparam()*, *pthread_attr_getschedpolicy()*,
 10883 *pthread_attr_getscope()*, *pthread_attr_getstackaddr()*, *pthread_attr_getstacksize()*,
 10884 *pthread_attr_setschedparam()*, *pthread_barrier_init()*, *pthread_barrierattr_getpshared()*,
 10885 *pthread_cond_init()*, *pthread_cond_signal()*, *pthread_cond_timedwait()*, *pthread_cond_wait()*,
 10886 *pthread_condattr_getclock()*, *pthread_condattr_getpshared()*, *pthread_create()*,
 10887 *pthread_getschedparam()*, *pthread_mutex_getprioceiling()*, *pthread_mutex_init()*,
 10888 *pthread_mutex_setprioceiling()*, *pthread_mutexattr_getprioceiling()*, *pthread_mutexattr_getprotocol()*,
 10889 *pthread_mutexattr_getpshared()*, *pthread_mutexattr_gettype()*, *pthread_rwlock_init()*,
 10890 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_rwlockattr_getpshared()*, and
 10891 *pthread_sigmask()*.

10892 **NAME**

10893 pwd.h — password structure

10894 **SYNOPSIS**

10895 #include <pwd.h>

10896 **DESCRIPTION**10897 The <pwd.h> header shall provide a definition for **struct passwd**, which shall include at least the
10898 following members:

10899	char	*pw_name	User's login name.
10900	uid_t	pw_uid	Numerical user ID.
10901	gid_t	pw_gid	Numerical group ID.
10902	char	*pw_dir	Initial working directory.
10903	char	*pw_shell	Program to use as shell.

10904 The **gid_t** and **uid_t** types shall be defined as described in <sys/types.h>.10905 The following shall be declared as functions and may also be defined as macros. Function
10906 prototypes shall be provided for use with an ISO C standard compiler.

```

10907 struct passwd *getpwnam(const char *);
10908 struct passwd *getpwuid(uid_t);
10909 TSF int getpwnam_r(const char *, struct passwd *, char *,
10910                size_t, struct passwd **);
10911 int getpwuid_r(uid_t, struct passwd *, char *,
10912               size_t, struct passwd **);
10913 XSI void endpwent(void);
10914 struct passwd *getpwent(void);
10915 void setpwent(void);

```

10916

10917 **APPLICATION USAGE**

10918 None.

10919 **RATIONALE**

10920 None.

10921 **FUTURE DIRECTIONS**

10922 None.

10923 **SEE ALSO**10924 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *endpwent()*, *getpwnam()*,
10925 *getpwuid()*10926 **CHANGE HISTORY**

10927 First released in Issue 1.

10928 **Issue 4**10929 Reference to the <sys/types.h> header is added for the definitions of **gid_t** and **uid_t**. This is
10930 marked as an extension.

10931 The following change is incorporated for alignment with the ISO POSIX-1 standard:

10932

- The function declarations in this header are expanded to full ISO C standard prototypes.

10933 **Issue 4, Version 2**

10934 For X/OPEN UNIX conformance, the *getpwent()*, *endpwent()*, and *setpwent()* functions are added
10935 to the list of functions declared in this header.

10936 **Issue 5**

10937 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

10938 **Issue 6**

10939 The following new requirements on POSIX implementations derive from alignment with the
10940 Single UNIX Specification:

- 10941 • The **gid_t** and **uid_t** types are mandated.
- 10942 • The *getpwnam_r()* and *getpwuid_r()* functions are marked as part of the
10943 `_POSIX_THREAD_SAFE_FUNCTIONS` option.

10944 **NAME**

10945 regex.h — regular expression matching types

10946 **SYNOPSIS**

10947 #include <regex.h>

10948 **DESCRIPTION**

10949 The <regex.h> header shall define the structures and symbolic constants used by the *regcomp()*,
10950 *regexexec()*, *regerror()*, and *regfree()* functions.

10951 The structure type **regex_t** shall contain at least the following member:

10952 size_t re_nsub Number of parenthesized subexpressions.

10953 The type **regoff_t** shall be defined as a signed arithmetic type that can hold the largest value that
10954 can be stored in either a type **off_t** or type **ssize_t**. The structure type **regmatch_t** shall contain
10955 at least the following members:

10956 regoff_t rm_so Byte offset from start of string
10957 to start of substring.

10958 regoff_t rm_eo Byte offset from start of string of the
10959 first character after the end of substring.

10960 Values for the *cflags* parameter to the *regcomp()* function:

10961 REG_EXTENDED Use Extended Regular Expressions.

10962 REG_ICASE Ignore case in match.

10963 REG_NOSUB Report only success or fail in *regexexec()*.

10964 REG_NEWLINE Change the handling of newline.

10965 Values for the *eflags* parameter to the *regexexec()* function:

10966 REG_NOTBOL The circumflex character ('^'), when taken as a special character, does
10967 not match the beginning of *string*.

10968 REG_NOTEOL The dollar sign ('\$'), when taken as a special character, does not match
10969 the end of *string*.

10970 The following constants shall be defined as error return values:

10971 REG_NOMATCH *regexexec()* failed to match.

10972 REG_BADPAT Invalid regular expression.

10973 REG_ECOLLATE Invalid collating element referenced.

10974 REG_ECTYPE Invalid character class type referenced.

10975 REG_EESCAPE Trailing '\\' in pattern.

10976 REG_ESUBREG Number in *\digit* invalid or in error.

10977 REG_EBRACK "[]" imbalance.

10978 REG_EPAREN " \()" or " ()" imbalance.

10979 REG_EBRACE " \{ \} " imbalance.

10980 REG_BADBR Content of " \{ \} " invalid: not a number, number too large, more than
10981 two numbers, first larger than second.

10982	REG_ERANGE	Invalid endpoint in range expression.
10983	REG_ESPACE	Out of memory.
10984	REG_BADRPT	'?', '*', or '+' not preceded by valid regular expression.
10985	REG_ENOSYS	The implementation does not support the function. (LEGACY)
10986	The following shall be declared as functions and may also be declared as macros. Function	
10987	prototypes shall be provided for use with an ISO C standard compiler.	
10988	<code>int regcomp(regex_t *restrict, const char *restrict, int);</code>	
10989	<code>size_t regerror(int, const regex_t *restrict, char *restrict, size_t);</code>	
10990	<code>int regexec(const regex_t *restrict, const char *restrict, size_t,</code>	
10991	<code>regmatch_t[restrict], int);</code>	
10992	<code>void regfree(regex_t *);</code>	
10993	The implementation may define additional macros or constants using names beginning with	
10994	REG_.	
10995	APPLICATION USAGE	
10996	None.	
10997	RATIONALE	
10998	None.	
10999	FUTURE DIRECTIONS	
11000	None.	
11001	SEE ALSO	
11002	The System Interfaces volume of IEEE Std. 1003.1-200x, <i>regcomp()</i> , the Shell and Utilities volume	
11003	of IEEE Std. 1003.1-200x	
11004	CHANGE HISTORY	
11005	First released in Issue 4.	
11006	Originally derived from the ISO POSIX-2 standard.	
11007	Issue 6	
11008	The REG_ENOSYS constant is marked LEGACY.	
11009	The restrict keyword is added to the prototypes for <i>regcomp()</i> , <i>regerror()</i> , and <i>regexec()</i> .	

11010 **NAME**

11011 sched.h — execution scheduling (**REALTIME**)

11012 **SYNOPSIS**

11013 PS #include <sched.h>

11014

11015 **DESCRIPTION**

11016 The <sched.h> header shall define the **sched_param** structure, which contains the scheduling
 11017 parameters required for implementation of each supported scheduling policy. This structure
 11018 shall contain at least the following member:

11019 int sched_priority Process execution scheduling priority.

11020 SS|TSP In addition, if **_POSIX_SPORADIC_SERVER** or **_POSIX_THREAD_SPORADIC_SERVER** is
 11021 defined, the **sched_param** structure defined in <sched.h> shall contain the following members
 11022 in addition to those specified above:

11023	int	sched_ss_low_priority	Low scheduling priority for
11024			sporadic server.
11025	struct timespec	sched_ss_repl_period	Replenishment period for
11026			sporadic server.
11027	struct timespec	sched_ss_init_budget	Initial budget for sporadic server.
11028	int	sched_ss_max_repl	Maximum pending replenishments for
11029			sporadic server.

11030

11031 Each process is controlled by an associated scheduling policy and priority. Associated with each
 11032 policy is a priority range. Each policy definition specifies the minimum priority range for that
 11033 policy. The priority ranges for each policy may overlap the priority ranges of other policies.

11034 Four scheduling policies are defined; others may be defined by the implementation. The four
 11035 standard policies are indicated by the values of the following symbolic constants:

11036 **SCHED_FIFO** First in-first out (FIFO) scheduling policy.

11037 **SCHED_RR** Round robin scheduling policy.

11038 SS|TSP **SCHED_SPORADIC** Sporadic server scheduling policy.

11039 **SCHED_OTHER** Another scheduling policy.

11040 The values of these constants are distinct.

11041 The following shall be declared as functions and may also be declared as macros. Function
 11042 prototypes shall be provided for use with an ISO C standard compiler.

```

11043 int sched_get_priority_max(int);
11044 int sched_get_priority_min(int);
11045 int sched_getparam(pid_t, struct sched_param *);
11046 int sched_getscheduler(pid_t);
11047 int sched_rr_get_interval(pid_t, struct timespec *);
11048 int sched_setparam(pid_t, const struct sched_param *);
11049 int sched_setscheduler(pid_t, int, const struct sched_param *);
11050 int sched_yield(void);
    
```

11051 Inclusion of the <sched.h> header makes symbols defined in the header <time.h> visible.

11052 **APPLICATION USAGE**

11053 None.

11054 **RATIONALE**

11055 None.

11056 **FUTURE DIRECTIONS**

11057 None.

11058 **SEE ALSO**11059 **<time.h>**11060 **CHANGE HISTORY**

11061 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

11062 **Issue 6**11063 The **<sched.h>** header is marked as part of the Process Scheduling option.11064 Sporadic server members are added to the **sched_param** structure, and the SCHED_SPORADIC
11065 scheduling policy is added for alignment with IEEE Std. 1003.1d-1999.11066 IEEE PASC Interpretation 1003.1 #108 is applied, correcting the **sched_param** structure whose
11067 members *sched_ss_repl_period* and *sched_ss_init_budget* members should be type **struct timespec**
11068 and not **timespec**.

11069 **NAME**

11070 search.h — search tables

11071 **SYNOPSIS**

11072 XSI #include <search.h>

11073

11074 **DESCRIPTION**

11075 The <search.h> header shall provide a type definition, **ENTRY**, for structure **entry** which shall
 11076 include the following members:

```
11077 char    *key
11078 void    *data
```

11079 and shall define **ACTION** and **VISIT** as enumeration data types through type definitions as
 11080 follows:

```
11081 enum { FIND, ENTER } ACTION;
11082 enum { preorder, postorder, endorder, leaf } VISIT;
```

11083 The **size_t** type shall be defined as described in <sys/types.h>.

11084 Each of the following shall be declared as a function, or defined as a macro, or both. Function
 11085 prototypes shall be provided for use with an ISO C standard compiler.

```
11086 int    hcreate(size_t);
11087 void   hdestroy(void);
11088 ENTRY *hsearch(ENTRY, ACTION);
11089 void   insque(void *, void *);
11090 void   *lfind(const void *, const void *, size_t *,
11091             size_t, int (*)(const void *, const void *));
11092 void   *lsearch(const void *, void *, size_t *,
11093             size_t, int (*)(const void *, const void *));
11094 void   remque(void *);
11095 void   *tdelete(const void *restrict, void **restrict,
11096             int (*)(const void *, const void *));
11097 void   *tfind(const void *, void *const *,
11098             int (*)(const void *, const void *));
11099 void   *tsearch(const void *, void **,
11100             int (*)(const void *, const void *));
11101 void   twalk(const void *,
11102             void (*)(const void *, VISIT, int ));
```

11103 **APPLICATION USAGE**

11104 None.

11105 **RATIONALE**

11106 None.

11107 **FUTURE DIRECTIONS**

11108 None.

11109 **SEE ALSO**

11110 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *hcreate()*, *insque()*,
 11111 *lsearch()*, *remque()*, *tsearch()*

11112 **CHANGE HISTORY**

11113 First released in Issue 1. Derived from Issue 1 of the SVID.

11114 **Issue 4**

11115 The function declarations in this header are expanded to full ISO C standard prototypes.

11116 Reference to the <sys/types.h> header is added for the definition of **size_t**.

11117 **Issue 4, Version 2**

11118 For X/OPEN UNIX conformance, the *insque()* and *remque()* functions are added to the list of
11119 functions declared in this header.

11120 **Issue 6**

11121 The Open Group corrigenda item U021/6 has been applied updating the prototypes for *tdelete()*
11122 and *tsearch()*.

11123 The **restrict** keyword is added to the prototype for *tdelete()*.

11124 **NAME**

11125 semaphore.h — semaphores (**REALTIME**)

11126 **SYNOPSIS**

11127 SEM `#include <semaphore.h>`

11128

11129 **DESCRIPTION**

11130 The <semaphore.h> header shall define the **sem_t** type, used in performing semaphore
 11131 operations. The semaphore may be implemented using a file descriptor, in which case
 11132 applications are able to open up at least a total of {OPEN_MAX} files and semaphores. The
 11133 symbol SEM_FAILED shall be defined (see *sem_open()*).

11134 The following shall be declared as functions and may also be declared as macros. Function
 11135 prototypes shall be provided for use with an ISO C standard compiler.

```

11136 int    sem_close(sem_t *);
11137 int    sem_destroy(sem_t *);
11138 int    sem_getvalue(sem_t *restrict, int *restrict);
11139 int    sem_init(sem_t *, int, unsigned);
11140 sem_t *sem_open(const char *, int, ...);
11141 int    sem_post(sem_t *);
11142 TMO   int    sem_timedwait(sem_t *restrict, const struct timespec *restrict);
11143 int    sem_trywait(sem_t *);
11144 int    sem_unlink(const char *);
11145 int    sem_wait(sem_t *);
    
```

11146 Inclusion of the <semaphore.h> header may make visible symbols defined in the headers
 11147 <fcntl.h> and <sys/types.h>.

11148 **APPLICATION USAGE**

11149 None.

11150 **RATIONALE**

11151 None.

11152 **FUTURE DIRECTIONS**

11153 None.

11154 **SEE ALSO**

11155 <fcntl.h>, <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *sem_destroy()*,
 11156 *sem_getvalue()*, *sem_init()*, *sem_open()*, *sem_post()*, *sem_timedwait()*, *sem_trywait()*, *sem_unlink()*,
 11157 *sem_wait()*

11158 **CHANGE HISTORY**

11159 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

11160 **Issue 6**

11161 The <semaphore.h> header is marked as part of the Semaphores option.

11162 The Open Group corrigenda item U021/3 has been applied, adding a description of
 11163 SEM_FAILED.

11164 The *sem_timedwait()* function is added for alignment with IEEE Std. 1003.1d-1999.

11165 The **restrict** keyword is added to the prototypes for *sem_getvalue()* and *sem_timedwait()*.

11166 **NAME**

11167 setjmp.h — stack environment declarations

11168 **SYNOPSIS**

11169 #include <setjmp.h>

11170 **DESCRIPTION**

11171 cx The functionality described on this reference page extends the ISO C standard. Applications
11172 shall define the appropriate feature test macro (see the System Interfaces volume of
11173 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
11174 symbols in this header.

11175 The **<setjmp.h>** header shall contain the type definitions for array types **jmp_buf** and
11176 **sigjmp_buf**.

11177 The following shall be declared as functions and may also be defined as macros. Function
11178 prototypes shall be provided for use with an ISO C standard compiler.

```
11179       void   longjmp(jmp_buf, int);  
11180       void   siglongjmp(sigjmp_buf, int);  
11181 xSI       void  _longjmp(jmp_buf, int);
```

11182

11183 Each of the following may be declared as a function, or defined as a macro, or both. Function
11184 prototypes shall be provided for use with an ISO C standard compiler.

```
11185       int    setjmp(jmp_buf);  
11186       int    sigsetjmp(sigjmp_buf, int);  
11187 xSI       int  _setjmp(jmp_buf);
```

11188

11189 **APPLICATION USAGE**

11190 None.

11191 **RATIONALE**

11192 None.

11193 **FUTURE DIRECTIONS**

11194 None.

11195 **SEE ALSO**

11196 The System Interfaces volume of IEEE Std. 1003.1-200x, *longjmp()*, *_longjmp()*, *setjmp()*,
11197 *siglongjmp()*, *sigsetjmp()*

11198 **CHANGE HISTORY**

11199 First released in Issue 1.

11200 **Issue 4**

11201 The following changes are incorporated for alignment with the ISO C standard:

- 11202 • The function declarations in this header are expanded to full ISO C standard prototypes.
- 11203 • The DESCRIPTION is changed to indicate that all functions in this header can also be
11204 declared as macros.
- 11205 • The arguments **jmp_buf** and **sigjmp_buf** are specified as array types.

11206 **Issue 4, Version 2**

11207 For X/OPEN UNIX conformance, the *_longjmp()* and *_setjmp()* functions are added to the list of
11208 functions declared in this header.

11209 NAME

11210 signal.h — signals

11211 SYNOPSIS

11212 #include <signal.h>

11213 DESCRIPTION

11214 cx The functionality described on this reference page extends the ISO C standard. Applications
11215 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
11217 symbols in this header.

11218 The <signal.h> header shall define the following symbolic constants, each of which expands to a
11219 distinct constant expression of the type:

11220 void (*)(int)

11221 whose value matches no declarable function.

11222 SIG_DFL Request for default signal handling.

11223 SIG_ERR Return value from *signal()* in case of error.

11224 SIG_HOLD Request that signal be held.

11225 SIG_IGN Request that signal be ignored.

11226 The following data types shall be defined through **typedef**:

11227 **sig_atomic_t** Possibly volatile-qualified integer type of an object that can be accessed as
11228 an atomic entity, even in the presence of asynchronous interrupts.

11229 **sigset_t** Integer or structure type of an object used to represent sets of signals.

11230 **pid_t** As described in <sys/types.h>.

11231 RTS The <signal.h> header shall define the **sigevent** structure, which has at least the following
11232 members:

11233	int	sigev_notify	Notification type.
11234	int	sigev_signo	Signal number.
11235	union sigval	sigev_value	Signal value.
11236	void (*)(union sigval)	sigev_notify_function	Notification function.
11237	(pthread_attr_t *)	sigev_notify_attributes	Notification attributes.

11238 The following values of *sigev_notify* shall be defined:

11239 SIGEV_NONE No asynchronous notification is delivered when the event of interest
11240 occurs.

11241 SIGEV_SIGNAL A queued signal, with an application-defined value, is generated when
11242 the event of interest occurs.

11243 SIGEV_THREAD A notification function is called to perform notification.

11244 The **sigval** union shall be defined as:

11245	int	sival_int	Integer signal value.
11246	void *	sival_ptr	Pointer signal value.

11247 This header shall also declare the macros SIGRTMIN and SIGRTMAX, which evaluate to
11248 integral expressions and, if the Realtime Signals Extension option is supported, specify a range
11249 of signal numbers that are reserved for application use and for which the realtime signal

11250 behavior specified in this volume of IEEE Std. 1003.1-200x is supported. The signal numbers in
 11251 this range do not overlap any of the signals specified in the following table.

11252 The range SIGRTMIN through SIGRTMAX inclusive shall include at least {RTSIG_MAX} signal
 11253 numbers.

11254 It is implementation-defined whether realtime signal behavior is supported for other signals.

11255 This header also declares the constants that are used to refer to the signals that occur in the
 11256 system. Signals defined here begin with the letters SIG. Each of the signals have distinct positive
 11257 integral values. The value 0 is reserved for use as the null signal (see *kill()*). Additional
 11258 implementation-defined signals may occur in the system.

11259 The following signals shall be supported on all implementations (default actions are explained
 11260 below the table):

11261

11262

	Signal	Default Action	Description
11263	SIGABRT	A	Process abort signal.
11264	SIGALRM	T	Alarm clock.
11265	SIGBUS	A	Access to an undefined portion of a memory object.
11266	SIGCHLD	I	Child process terminated or stopped.
11267	SIGCONT	C	Continue executing, if stopped.
11268	SIGFPE	A	Erroneous arithmetic operation.
11269	SIGHUP	T	Hangup.
11270	SIGILL	A	Illegal instruction.
11271	SIGINT	T	Terminal interrupt signal.
11272	SIGKILL	T	Kill (cannot be caught or ignored).
11273	SIGPIPE	T	Write on a pipe with no one to read it.
11274	SIGQUIT	A	Terminal quit signal.
11275	SIGSEGV	A	Invalid memory reference.
11276	SIGSTOP	S	Stop executing (cannot be caught or ignored).
11277	SIGTERM	T	Termination signal.
11278	SIGTSTP	S	Terminal stop signal.
11279	SIGTTIN	S	Background process attempting read.
11280	SIGTTOU	S	Background process attempting write.
11281	SIGUSR1	T	User-defined signal 1.
11282	SIGUSR2	T	User-defined signal 2.
11283 XSI	SIGPOLL	T	Pollable event.
11284	SIGPROF	T	Profiling timer expired.
11285	SIGSYS	A	Bad system call.
11286	SIGTRAP	A	Trace/breakpoint trap.
11287	SIGURG	I	High bandwidth data is available at a socket.
11288 XSI	SIGVTALRM	T	Virtual timer expired.
11289	SIGXCPU	A	CPU time limit exceeded.
11290	SIGXFSZ	A	File size limit exceeded.

11291 The default actions are as follows:

11292 T Abnormal termination of the process. The process is terminated with all the consequences
 11293 of *_exit()* except that the status made available to *wait()* and *waitpid()* indicates abnormal
 11294 termination by the specified signal.

11295 A Abnormal termination of the process.

11296 XSI Additionally, implementation-defined abnormal termination actions, such as creation of a
 11297 core file, may occur.

11298 I Ignore the signal.
11299 S Stop the process.
11300 C Continue the process, if it is stopped; otherwise, ignore the signal.

11301 The header shall provide a declaration of **struct sigaction**, including at least the following
11302 members:

11303 void (*sa_handler)(int) What to do on receipt of signal.
11304 sigset_t sa_mask Set of signals to be blocked during execution
11305 of the signal handling function.
11306 int sa_flags Special flags.
11307 void (*)(int, siginfo_t *, void *) sa_sigaction
11308 Pointer to signal handler function or one
11309 of the macros SIG_IGN or SIG_DFL.

11310 XSI The storage occupied by *sa_handler* and *sa_sigaction* may overlap, and a portable program must
11311 not use both simultaneously.

11312 The following shall be declared as constants:

11313 SA_NOCLDSTOP Do not generate SIGCHLD when children stop.
11314 SIG_BLOCK The resulting set is the union of the current set and the signal set pointed
11315 to by the argument *set*.
11316 SIG_UNBLOCK The resulting set is the intersection of the current set and the complement
11317 of the signal set pointed to by the argument *set*.
11318 SIG_SETMASK The resulting set is the signal set pointed to by the argument *set*.

11319 XSI SA_ONSTACK Causes signal delivery to occur on an alternate stack.
11320 XSI SA_RESETHAND Causes signal dispositions to be set to SIG_DFL on entry to signal
11321 handlers.
11322 XSI SA_RESTART Causes certain functions to become restartable.
11323 XSI SA_SIGINFO Causes extra information to be passed to signal handlers at the time of
11324 receipt of a signal.
11325 XSI SA_NOCLDWAIT Causes implementations not to create zombie processes on child death.
11326 XSI SA_NODEFER Causes signal not to be automatically blocked on entry to signal handler.
11327 XSI SS_ONSTACK Process is executing on an alternate signal stack.
11328 XSI SS_DISABLE Alternate signal stack is disabled.
11329 XSI MINSIGSTKSZ Minimum stack size for a signal handler.
11330 XSI SIGSTKSZ Default size in bytes for the alternate signal stack.

11331 XSI The **ucontext_t** structure shall be defined through **typedef** as described in <ucontext.h>.
11332 The **mcontext_t** type shall be defined through **typedef** as described in <ucontext.h>.
11333 The <signal.h> header shall define the **stack_t** type as a structure that includes at least the
11334 following members:

11335 void *ss_sp Stack base or pointer.
11336 size_t ss_size Stack size.
11337 int ss_flags Flags.

11338 The <signal.h> header shall define the **sigstack** structure that includes at least the following
 11339 members:

11340	int	ss_onstack	Non-zero when signal stack is in use.
11341	void	*ss_sp	Signal stack pointer.

11342

11343 The <signal.h> header shall define the **siginfo_t** type as a structure that includes at least the
 11344 following members:

11345	int	si_signo	Signal number.
11346 XSI	int	si_errno	If non-zero, an <i>errno</i> value associated with 11347 this signal, as defined in <errno.h>.
11348	int	si_code	Signal code.
11349 XSI	pid_t	si_pid	Sending process ID.
11350	uid_t	si_uid	Real user ID of sending process.
11351	void	*si_addr	Address of faulting instruction.
11352	int	si_status	Exit value or signal.
11353	long	si_band	Band event for SIGPOLL.
11354 RTS	union sigval	si_value	Signal value.

11355

11356 The macros specified in the **Code** column of the following table are defined for use as values of
 11357 XSI *si_code* that are signal-specific or non-signal-specific reasons why the signal was generated.

11358

11359

11360 XSI

11361

11362

11363

11364

11365

11366

11367

11368

11369

11370

11371

11372

11373

11374

11375

11376

11377

11378

11379

11380

11381

11382

11383

11384

11385

11386

11387

11388

11389

11390

11391

11392

11393

11394

11395

11396

11397

11398

11399

11400

11401

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.

11402 XSI

11403

11404

11405

11406

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.

11407 In addition, the following signal-specific information shall be available:

11408

11409

11410

11411

11412

11413

11414

11415

11416

11417

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.

11418

For some implementations, the value of *si_addr* may be inaccurate.

11419

The following shall be declared as functions and may also be defined as macros:

11420 XSI

```
void (*bsd_signal(int, void (*)(int)))(int);
```

11421

```
int kill(pid_t, int);
```

11422 XSI

```
int killpg(pid_t, int);
```

11423

```
int pthread_kill(pthread_t, int);
```

11424

```
int pthread_sigmask(int, const sigset_t *, sigset_t *);
```

11425

```
int raise(int);
```

11426

```
int sigaction(int, const struct sigaction *restrict,  
             struct sigaction *restrict);
```

11427

```
int sigaddset(sigset_t *, int);
```

11429 XSI

```
int sigaltstack(const stack_t *restrict, stack_t *restrict);
```

11430

```
int sigdelset(sigset_t *, int);
```

11431

```
int sigemptyset(sigset_t *);
```

11432

```
int sigfillset(sigset_t *);
```

11433 XSI

```
int sighold(int);
```

11434

```
int sigignore(int);
```

11435

```
int siginterrupt(int, int);
```

11436

```
int sigismember(const sigset_t *, int);
```

11437

```
void (*signal(int, void (*)(int)))(int);
```

11438 XSI

```
int sigpause(int);
```

11439

```
int sigpending(sigset_t *);
```

11440

```
int sigprocmask(int, const sigset_t *restrict, sigset_t *restrict);
```

11441 RTS

```
int sigqueue(pid_t, int, const union sigval);
```

11442 XSI

```
int sigrelse(int);
```

11443

```
void (*sigset(int, void (*)(int)))(int);
```

11444

```
int sigstack(struct sigstack *, struct sigstack *); (LEGACY)
```

11445

```
int sigsuspend(const sigset_t *);
```

11446 RTS

```
int sigtimedwait(const sigset_t *restrict, siginfo_t *restrict,  
               const struct timespec *restrict);
```

11447

```
int sigwait(const sigset_t *restrict, int *restrict);
```

11449 RTS

```
int sigwaitinfo(const sigset_t *restrict, siginfo_t *restrict);
```

11450

11451 **APPLICATION USAGE**

11452 None.

11453 **RATIONALE**

11454 None.

11455 **FUTURE DIRECTIONS**

11456 None.

11457 **SEE ALSO**

11458 **<errno.h>**, **<stropts.h>**, **<sys/types.h>**, **<ucontext.h>**, the System Interfaces volume of
 11459 IEEE Std. 1003.1-200x, *alarm()*, *bsd_signal()*, *ioctl()*, *kill()*, *killpg()*, *raise()*, *sigaction()*, *sigaddset()*,
 11460 *sigaltstack()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *siginterrupt()*, *sigismember()*, *signal()*,
 11461 *sigpending()*, *sigprocmask()*, *sigqueue()*, *sigsuspend()*, *sigwaitinfo()*, *wait()*, *waitid()*

11462 **CHANGE HISTORY**

11463 First released in Issue 1.

11464 **Issue 4**11465 A reference to **<sys/types.h>** is added for the definition of **pid_t**. This is marked as an extension.

11466 In the list of signals starting with SIGCHLD, the statement “but a system not supporting the job
 11467 control option is not obliged to support the functionality of these signals” is removed. This is
 11468 because job control is defined as mandatory on Issue 4 conforming implementations.

11469 Reference to implementation-defined abnormal termination routines, such as creation of a core
 11470 file, in item ii in the defaults action list is marked as an extension.

11471 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 11472 • The function declarations in this header are expanded to full ISO C standard prototypes.
- 11473 • The DESCRIPTION is changed as follows:
 - 11474 — To define the type **sig_atomic_t**.
 - 11475 — To define the syntax of signal names and functions.
 - 11476 — To combine the two tables of constants.
 - 11477 — SIGFPE is no longer limited to floating-point exceptions, but covers all erroneous
 11478 arithmetic operations.

11479 The following change is incorporated for alignment with the ISO C standard:

- 11480 • The *raise()* function is added to the list of functions declared in this header.

11481 **Issue 4, Version 2**

11482 The following changes are incorporated for X/OPEN UNIX conformance:

- 11483 • The SIGTRAP, SIGBUS, SIGSYS, SIGPOLL, SIGPROF, SIGXCPU, SIGXFSZ, SIGURG, and
 11484 SIGVTALRM signals are added to the list of signals that are supported on all conforming
 11485 implementations.
- 11486 • The *sa_sigaction* member is added to the **sigaction** structure, and a note is added that the
 11487 storage used by *sa_handler* and *sa_sigaction* may overlap.
- 11488 • The SA_ONSTACK, SA_RESETHAND, SA_RESTART, SA_SIGINFO, SA_NOCLDWAIT,
 11489 SS_ONSTACK, SS_DISABLE, MINSIGSTKSZ, and SIGSTKSZ constants are defined. The
 11490 **stack_t**, **sigstack**, and **siginfo** structures are defined.
- 11491 • Definitions are given for the **ucontext_t**, **stack_t**, **sigstack**, and **siginfo_t** types.

- 11492 • A table is provided listing macros that are defined as signal-specific reasons why a signal
11493 was generated. Signal-specific additional information is specified.
- 11494 • The *bsd_signal()*, *killpg()*, *_longjmp()*, *_setjmp()*, *sigaltstack()*, *sighold()*, *sigignore()*,
11495 *siginterrupt()*, *sigpause()*, *sigrelse()*, *sigset()*, and *sigstack()* functions are added to the list of
11496 functions declared in this header.
- 11497 **Issue 5**
- 11498 The DESCRIPTION is updated for alignment with POSIX Realtime Extension and the POSIX
11499 Threads Extension.
- 11500 The default action for SIGURG is changed for i to iii. The function prototype for *sigmask()* is
11501 removed.
- 11502 **Issue 6**
- 11503 The Open Group corrigenda item U035/2 has been applied. In the DESCRIPTION, the wording
11504 for abnormal termination is clarified.
- 11505 The Open Group corrigenda item U028/8 has been applied, correcting the prototype for the
11506 *sigset()* function.
- 11507 The Open Group corrigenda item U026/3 has been applied, correcting the type of the
11508 *sigev_notify_function* function member of the **sigevent** structure.
- 11509 The following new requirements on POSIX implementations derive from alignment with the
11510 Single UNIX Specification:
- 11511 • The SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, and SIGTTOU signals are now
11512 mandated. This is also a FIPS requirement.
- 11513 • The **pid_t** definition is mandated.
- 11514 The RT markings are now changed to RTS to denote that the semantics are part of the Realtime
11515 Signals Extension option.
- 11516 The **restrict** keyword is added to the prototypes for *sigaction()*, *sigaltstack()*, *sigprocmask()*,
11517 *sigtimedwait()*, *sigwait()*, and *sigwaitinfo()*.

11518 **NAME**11519 spawn.h — spawn (**REALTIME**)11520 **SYNOPSIS**

11521 SPN #include <spawn.h>

11522

11523 **DESCRIPTION**11524 The <spawn.h> header shall define the **posix_spawnattr_t** and **posix_spawn_file_actions_t**
11525 types used in performing spawn operations.11526 The <spawn.h> header shall define the flags that may be set in a **posix_spawnattr_t** object using
11527 the *posix_spawnattr_setflags()* function:

11528 POSIX_SPAWN_RESETEIDS

11529 POSIX_SPAWN_SETPGROUP

11530 PS POSIX_SPAWN_SETSCHEDPARAM

11531 POSIX_SPAWN_SETSCHEDULER

11532 POSIX_SPAWN_SETSIGDEF

11533 POSIX_SPAWN_SETSIGMASK

11534 The following shall be declared as functions and may also be declared as macros. Function
11535 prototypes shall be provided for use with an ISO C standard compiler.

```

11536 int    posix_spawn(pid_t *restrict, const char *restrict,
11537                  const posix_spawn_file_actions_t *,
11538                  const posix_spawnattr_t *restrict, char *const [restrict],
11539                  char *const [restrict]);
11540 int    posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *,
11541                  int);
11542 int    posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *,
11543                  int, int);
11544 int    posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *restrict,
11545                  int, const char *restrict, int, mode_t);
11546 int    posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *);
11547 int    posix_spawn_file_actions_init(posix_spawn_file_actions_t *);
11548 int    posix_spawnattr_destroy(posix_spawnattr_t *);
11549 int    posix_spawnattr_getsigdefault(const posix_spawnattr_t *restrict,
11550                  sigset_t *restrict);
11551 int    posix_spawnattr_getflags(const posix_spawnattr_t *restrict,
11552                  short *restrict);
11553 int    posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict,
11554                  pid_t *restrict);
11555 PS int    posix_spawnattr_getschedparam(const posix_spawnattr_t *restrict,
11556                  struct sched_param *restrict);
11557 int    posix_spawnattr_getschedpolicy(const posix_spawnattr_t *restrict,
11558                  int *restrict);
11559 int    posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict,
11560                  sigset_t *restrict);
11561 int    posix_spawnattr_init(posix_spawnattr_t *);
11562 int    posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict,
11563                  const sigset_t *restrict);
11564 int    posix_spawnattr_setflags(posix_spawnattr_t *, short);
11565 int    posix_spawnattr_setpgroup(posix_spawnattr_t *, pid_t);

```

11566 PS

```

11567 int    posix_spawnattr_setschedparam(posix_spawnattr_t *restrict,
11568                                     const struct sched_param *restrict);
11569 int    posix_spawnattr_setschedpolicy(posix_spawnattr_t *, int);
11570 int    posix_spawnattr_setsigmask(posix_spawnattr_t *restrict,
11571                                   const sigset_t *restrict);
11572 const posix_spawnattr_t *, char *const [], char *const []);
11573 int    posix_spawn(pid_t *restrict, const char *restrict,
11574                  const posix_spawn_file_actions_t *,
11575                  const posix_spawnattr_t *restrict,
11576                  char *const [restrict], char *const [restrict]);

```

11577 Inclusion of the <spawn.h> header may make visible symbols defined in the <sched.h>, |
11578 <signal.h>, and <sys/types.h> headers. |

11579 **APPLICATION USAGE**

11580 None.

11581 **RATIONALE**

11582 None.

11583 **FUTURE DIRECTIONS**

11584 None.

11585 **SEE ALSO**

11586 <sched.h>, <semaphore.h>, <signal.h>, <sys/types.h>, the System Interfaces volume of |
11587 IEEE Std. 1003.1-200x, *posix_spawnattr_destroy()*, *posix_spawnattr_getsigdefault()*, |
11588 *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*, |
11589 *posix_spawnattr_getschedpolicy()*, *posix_spawnattr_getsigmask()*, *posix_spawnattr_init()*, |
11590 *posix_spawnattr_setsigdefault()*, *posix_spawnattr_setflags()*, *posix_spawnattr_setpgroup()*, |
11591 *posix_spawnattr_setschedparam()*, *posix_spawnattr_setschedpolicy()*, *posix_spawnattr_setsigmask()*, |
11592 *posix_spawn()*, *posix_spawn_file_actions_addclose()*, *posix_spawn_file_actions_adddup2()*, |
11593 *posix_spawn_file_actions_addopen()*, *posix_spawn_file_actions_destroy()*, |
11594 *posix_spawn_file_actions_init()*, *posix_spawnnp()*

11595 **CHANGE HISTORY**

11596 First released in Issue 6. Included for alignment with IEEE Std. 1003.1d-1999. |

11597 The **restrict** keyword is added to the prototypes for *posix_spawn()*, |
11598 *posix_spawn_file_actions_addopen()*, *posix_spawnattr_getsigdefault()*, *posix_spawnattr_getflags()*, |
11599 *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*, *posix_spawnattr_getschedpolicy()*, |
11600 *posix_spawnattr_getsigmask()*, *posix_spawnattr_setsigdefault()*, *posix_spawnattr_setschedparam()*, |
11601 *posix_spawnattr_setsigmask()*, and *posix_spawnnp()*.

11602 **NAME**11603 **stdarg.h** — handle variable argument list11604 **SYNOPSIS**

11605 #include <stdarg.h>

11606 void va_start(va_list *ap*, *argN*);11607 void va_copy(va_list *dest*, va_list *src*);11608 type va_arg(va_list *ap*, *type*);11609 void va_end(va_list *ap*);11610 **DESCRIPTION**

11611 **CX** The functionality described on this reference page extends the ISO C standard. Applications
11612 shall define the appropriate feature test macro (see the System Interfaces volume of
11613 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
11614 symbols in this header.

11615 The **<stdarg.h>** header contains a set of macros which allows portable functions that accept
11616 variable argument lists to be written. Functions that have variable argument lists (such as
11617 *printf()*) but do not use these macros, are inherently non-portable, as different systems use
11618 different argument-passing conventions.

11619 The type **va_list** is defined for variables used to traverse the list.

11620 The *va_start()* macro is invoked to initialize *ap* to the beginning of the list before any calls to
11621 *va_arg()*.

11622 The *va_copy()* macro initializes *dest* as a copy of *src*, as if the *va_start()* macro had been applied to
11623 *dest* followed by the same sequence of uses of the *va_arg()* macro as had previously been used to
11624 reach the present state of *src*. Neither the *va_copy()* nor *va_start()* macro shall be invoked to
11625 reinitialize *dest* without an intervening invocation of the *va_end()* macro for the same *dest*.

11626 The object *ap* may be passed as an argument to another function; if that function invokes the
11627 *va_arg()* macro with parameter *ap*, the value of *ap* in the calling function is indeterminate and
11628 must be passed to the *va_end()* macro prior to any further reference to *ap*. The parameter *argN* is
11629 the identifier of the rightmost parameter in the variable parameter list in the function definition
11630 (the one just before the ...). If the parameter *argN* is declared with the **register** storage class, with
11631 a function type or array type, or with a type that is not compatible with the type that results after
11632 application of the default argument promotions, the behavior is undefined.

11633 The *va_arg()* macro returns the next argument in the list pointed to by *ap*. Each invocation of
11634 *va_arg()* modifies *ap* so that the values of successive arguments are returned in turn. The *type*
11635 parameter is the type the argument is expected to be. This is the type name specified such that
11636 the type of a pointer to an object that has the specified type can be obtained simply by suffixing
11637 a '*' to type. Different types can be mixed, but it is up to the routine to know what type of
11638 argument is expected.

11639 The *va_end()* macro is used to clean up; it invalidates *ap* for use (unless *va_start()* or *va_copy()* is
11640 invoked again).

11641 Each invocation of the *va_start()* and *va_copy()* macros shall be matched by a corresponding
11642 invocation of the *va_end()* macro in the same function.

11643 Multiple traversals, each bracketed by *va_start()* ... *va_end()*, are possible.

11644 **EXAMPLES**

11645 This example is a possible implementation of *execl()*:

```
11646 #include <stdarg.h>
11647 #define MAXARGS 31
11648 /*
11649  * execl is called by
11650  * execl(file, arg1, arg2, ..., (char *) (0));
11651  */
11652 int execl(const char *file, const char *args, ...)
11653 {
11654     va_list ap;
11655     char *array[MAXARGS];
11656     int argno = 0;
11657     va_start(ap, args);
11658     while (args != 0) {
11659         array[argno++] = args;
11660         args = va_arg(ap, const char *);
11661     }
11662     va_end(ap);
11663     return execv(file, array);
11664 }
```

11665 **APPLICATION USAGE**

11666 It is up to the calling routine to communicate to the called routine how many arguments there
11667 are, since it is not always possible for the called routine to determine this in any other way. For
11668 example, *execl()* is passed a null pointer to signal the end of the list. The *printf()* function can tell
11669 how many arguments are there by the *format* argument.

11670 **RATIONALE**

11671 None.

11672 **FUTURE DIRECTIONS**

11673 None.

11674 **SEE ALSO**

11675 The System Interfaces volume of IEEE Std. 1003.1-200x, *exec()*, *printf()*

11676 **CHANGE HISTORY**

11677 First released in Issue 4. Derived from the ANSI C standard.

11678 **NAME**

11679 stdbool.h — boolean type and values

11680 **SYNOPSIS**

11681 #include <stdbool.h>

11682 **DESCRIPTION**

11683 cx The functionality described on this reference page extends the ISO C standard. Applications
11684 shall define the appropriate feature test macro (see the System Interfaces volume of
11685 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
11686 symbols in this header.

11687 The **<stdbool.h>** header shall define the following macros:11688 *bool* Expands to *_Bool*.11689 *true* Expands to the integer constant 1.11690 *false* Expands to the integer constant 0.11691 *__bool_true_false_are_defined*

11692 Expands to the integer constant 1.

11693 An application may undefine and then possibly redefine the macros *bool*, *true*, and *false*.11694 **APPLICATION USAGE**

11695 None.

11696 **RATIONALE**

11697 None.

11698 **FUTURE DIRECTIONS**

11699 The ability to undefine and redefine the macros *bool*, *true*, and *false* is an obsolescent feature and
11700 may be withdrawn in the future.

11701 **SEE ALSO**

11702 None.

11703 **CHANGE HISTORY**

11704 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

11705 **NAME**11706 **stddef.h** — standard type definitions11707 **SYNOPSIS**

11708 #include <stddef.h>

11709 **DESCRIPTION**

11710 **cx** The functionality described on this reference page extends the ISO C standard. Applications
11711 shall define the appropriate feature test macro (see the System Interfaces volume of
11712 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
11713 symbols in this header.

11714 The <stddef.h> header shall define the following:

11715 **NULL** Null pointer constant.11716 **offsetof(*type*, *member-designator*)**

11717 Integral constant expression of type **size_t**, the value of which is the offset in bytes
11718 to the structure member (*member-designator*), from the beginning of its structure
11719 (*type*).

11720 The <stddef.h> header shall define through **typedef**:11721 **ptrdiff_t** Signed integer type of the result of subtracting two pointers.

11722 **wchar_t** Integer type whose range of values can represent distinct wide-character codes for
11723 all members of the largest character set specified among the locales supported by
11724 the compilation environment: the null character has the code value 0 and each
11725 member of the Portable Character Set has a code value equal to its value when
11726 used as the lone character in an integer character constant.

11727 **size_t** Unsigned integer type of the result of the *sizeof* operator.11728 **APPLICATION USAGE**

11729 None.

11730 **RATIONALE**

11731 None.

11732 **FUTURE DIRECTIONS**

11733 None.

11734 **SEE ALSO**

11735 <wchar.h>, <sys/types.h>

11736 **CHANGE HISTORY**

11737 First released in Issue 4. Derived from the ANSI C standard.

11738 **NAME**

11739 stdint.h — integer types

11740 **SYNOPSIS**

11741 #include <stdint.h>

11742 **DESCRIPTION**

11743 **CX** The functionality described on this reference page extends the ISO C standard. Applications
11744 shall define the appropriate feature test macro (see the System Interfaces volume of
11745 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
11746 symbols in this header.

11747 The <stdint.h> header declares sets of integer types having specified widths, and defines
11748 corresponding sets of macros. It also defines macros that specify limits of integer types
11749 corresponding to types defined in other standard headers.

11750 Types are defined in the following categories:

- 11751 • Integer types having certain exact widths
- 11752 • Integer types having at least certain specified widths
- 11753 • Fastest integer types having at least certain specified widths
- 11754 • Integer types wide enough to hold pointers to objects
- 11755 • Integer types having greatest width

11756 (Some of these types may denote the same type.)

11757 Corresponding macros specify limits of the declared types and construct suitable constants.

11758 For each type described herein that the implementation provides, the <stdint.h> header shall
11759 declare that **typedef** name and define the associated macros. Conversely, for each type described
11760 herein that the implementation does not provide, the <stdint.h> header shall not declare that
11761 **typedef** name, nor shall it define the associated macros. An implementation shall provide those
11762 types described as required, but need not provide any of the others (described as optional).

11763 **Integer Types**

11764 When **typedef** names differing only in the absence or presence of the initial *u* are defined, they
11765 shall denote corresponding signed and unsigned types as described in the ISO/IEC 9899:1999
11766 standard, Section 6.2.5; an implementation providing one of these corresponding types shall also
11767 provide the other.

11768 In the following descriptions, the symbol *N* represents an unsigned decimal integer with no
11769 leading zeros (for example, 8 or 24, but not 04 or 048).

- 11770 • Exact-width integer types

11771 The **typedef** name **intN_t** designates a signed integer type with width *N*, no padding bits,
11772 and a two's-complement representation. Thus, **int8_t** denotes a signed integer type with a
11773 width of exactly 8 bits.

11774 The **typedef** name **uintN_t** designates an unsigned integer type with width *N*. Thus,
11775 **uint24_t** denotes an unsigned integer type with a width of exactly 24 bits.

11776 These types are optional. However, if an implementation provides integer types with widths
11777 of 8, 16, 32, or 64 bits, it shall define the corresponding **typedef** names.

- 11778 • Minimum-width integer types

11779 The **typedef** name **int_leastN_t** designates a signed integer type with a width of at least *N*,
 11780 such that no signed integer type with lesser size has at least the specified width. Thus,
 11781 **int_least32_t** denotes a signed integer type with a width of at least 32 bits.

11782 The **typedef** name **uint_leastN_t** designates an unsigned integer type with a width of at least
 11783 *N*, such that no unsigned integer type with lesser size has at least the specified width. Thus,
 11784 **uint_least16_t** denotes an unsigned integer type with a width of at least 16 bits.

11785 The following types are required:

```
11786     int_least8_t
11787     int_least16_t
11788     int_least32_t
11789     int_least64_t
11790     uint_least8_t
11791     uint_least16_t
11792     uint_least32_t
11793     uint_least64_t
```

11794 All other types of this form are optional.

- 11795 • Fastest minimum-width integer types

11796 Each of the following types designates an integer type that is usually fastest to operate with
 11797 among all integer types that have at least the specified width.

11798 The designated type is not guaranteed to be fastest for all purposes; if the implementation
 11799 has no clear grounds for choosing one type over another, it will simply pick some integer
 11800 type satisfying the signedness and width requirements.

11801 The **typedef** name **int_fastN_t** designates the fastest signed integer type with a width of at
 11802 least *N*. The **typedef** name **uint_fastN_t** designates the fastest unsigned integer type with a
 11803 width of at least *N*.

11804 The following types are required:

```
11805     int_fast8_t
11806     int_fast16_t
11807     int_fast32_t
11808     int_fast64_t
11809     uint_fast8_t
11810     uint_fast16_t
11811     uint_fast32_t
11812     uint_fast64_t
```

11813 All other types of this form are optional.

- 11814 • Integer types capable of holding object pointers

11815 The following type designates a signed integer type with the property that any valid pointer
 11816 to **void** can be converted to this type, then converted back to a pointer to **void**, and the result
 11817 will compare equal to the original pointer:

```
11818     intptr_t
```

11819 The following type designates an unsigned integer type with the property that any valid
 11820 pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and
 11821 the result will compare equal to the original pointer:

11822 **uintptr_t**

11823 These types are optional.

11824 • Greatest-width integer types

11825 The following type designates a signed integer type capable of representing any value of any

11826 signed integer type:

11827 **intmax_t**

11828 The following type designates an unsigned integer type capable of representing any value of

11829 any unsigned integer type:

11830 **uintmax_t**

11831 These types are required.

11832 **Limits of Specified-Width Integer Types**

11833 The following object-like macros specify the minimum and maximum limits of the types

11834 declared in the <stdint.h> header. Each macro name corresponds to a similar type name in

11835 **Integer Types** (on page 352).

11836 Each instance of any defined macro shall be replaced by a constant expression suitable for use in

11837 #if preprocessing directives, and this expression shall have the same type as would an

11838 expression that is an object of the corresponding type converted according to the integer

11839 promotions. Its implementation-defined value shall be equal to or greater in magnitude

11840 (absolute value) than the corresponding value given below, with the same sign, except where

11841 stated to be exactly the given value.

11842 • Limits of exact-width integer types

11843 — Minimum values of exact-width signed integer types:

11844 {INTN_MIN} Exactly $-(2^{N-1})$

11845 — Maximum values of exact-width signed integer types:

11846 {INTN_MAX} Exactly $2^{N-1} - 1$

11847 — Maximum values of exact-width unsigned integer types:

11848 {UINTN_MAX} Exactly $2^N - 1$

11849 • Limits of minimum-width integer types

11850 — Minimum values of minimum-width signed integer types:

11851 {INT_LEASTN_MIN} $-(2^{N-1} - 1)$

11852 — Maximum values of minimum-width signed integer types:

11853 {INT_LEASTN_MAX} $2^N - 1$

11854 — Maximum values of minimum-width unsigned integer types:

11855 {UINT_LEASTN_MAX} $2^N - 1$

11856 • Limits of fastest minimum-width integer types

11857 — Minimum values of fastest minimum-width signed integer types:

11858 {INT_FASTN_MIN} $-(2^{N-1} - 1)$

11859	—	Maximum values of fastest minimum-width signed integer types:	
11860		{INT_FASTN_MAX}	$2^{N-1} - 1$
11861	—	Maximum values of fastest minimum-width unsigned integer types:	
11862		{UINT_FASTN_MAX}	$2^N - 1$
11863	•	Limits of integer types capable of holding object pointers	
11864	—	Minimum value of pointer-holding signed integer type:	
11865		{INTPTR_MIN}	$-(2^{15} - 1)$
11866	—	Maximum value of pointer-holding signed integer type:	
11867		{INTPTR_MAX}	$2^{15} - 1$
11868	—	Maximum value of pointer-holding unsigned integer type:	
11869		{UINTPTR_MAX}	$2^{16} - 1$
11870	•	Limits of greatest-width integer types	
11871	—	Minimum value of greatest-width signed integer type:	
11872		{INTMAX_MIN}	$-(2^{63} - 1)$
11873	—	Maximum value of greatest-width signed integer type:	
11874		{INTMAX_MAX}	$2^{63} - 1$
11875	—	Maximum value of greatest-width unsigned integer type:	
11876		{UINTMAX_MAX}	$2^{64} - 1$
11877		Limits of Other Integer Types	
11878		The following object-like macros specify the minimum and maximum limits of integer types	
11879		corresponding to types defined in other standard headers.	
11880		Each instance of these macros shall be replaced by a constant expression suitable for use in #if	
11881		preprocessing directives, and this expression shall have the same type as would an expression	
11882		that is an object of the corresponding type converted according to the integer promotions. Its	
11883		implementation-defined value shall be equal to or greater in magnitude (absolute value) than	
11884		the corresponding value given below, with the same sign.	
11885	•	Limits of ptrdiff_t :	
11886		{PTRDIFF_MIN}	-65535
11887		{PTRDIFF_MAX}	+65535
11888	•	Limits of sig_atomic_t :	
11889		{SIG_ATOMIC_MIN}	See below.
11890		{SIG_ATOMIC_MAX}	See below.
11891	•	Limit of size_t :	
11892		{SIZE_MAX}	65535
11893	•	Limits of wchar_t :	
11894		{WCHAR_MIN}	See below.

11895 {WCHAR_MAX} See below.

11896 • Limits of **wint_t**:

11897 {WINT_MIN} See below.

11898 [WINT_MAX] See below.

11899 If **sig_atomic_t** (see the **<signal.h>** header) is defined as a signed integer type, the value of
 11900 {SIG_ATOMIC_MIN} shall be no greater than -127 and the value of {SIG_ATOMIC_MAX} shall
 11901 be no less than 127 ; otherwise, **sig_atomic_t** is defined as an unsigned integer type, and the
 11902 value of {SIG_ATOMIC_MIN} shall be 0 and the value of {SIG_ATOMIC_MAX} shall be no less
 11903 than 255 .

11904 If **wchar_t** (see the **<stddef.h>** header) is defined as a signed integer type, the value of
 11905 {WCHAR_MIN} shall be no greater than -127 and the value of {WCHAR_MAX} shall be no less
 11906 than 127 ; otherwise, **wchar_t** is defined as an unsigned integer type, and the value of
 11907 {WCHAR_MIN} shall be 0 and the value of {WCHAR_MAX} shall be no less than 255 .

11908 If **wint_t** (see the **<wchar.h>** header) is defined as a signed integer type, the value of
 11909 {WINT_MIN} shall be no greater than -32767 and the value of {WINT_MAX} shall be no less
 11910 than 32767 ; otherwise, **wint_t** is defined as an unsigned integer type, and the value of
 11911 {WINT_MIN} shall be 0 and the value of {WINT_MAX} shall be no less than 65535 .

11912 **Macros for Integer Constants**

11913 The following function-like macros expand to integer constants suitable for initializing objects
 11914 that have integer types corresponding to types defined in the **<stdint.h>** header. Each macro
 11915 name corresponds to a similar type name listed under *Minimum-width integer types* and *Greatest-*
 11916 *width integer types*.

11917 The argument in any instance of these macros shall be a decimal, octal, or hexadecimal constant
 11918 with a value that does not exceed the limits for the corresponding type.

11919 • Macros for minimum-width integer constants

11920 Each of the following macros expands to an integer constant having the value specified by its
 11921 argument and a type with at least the specified width.

11922 The macro *INTN_C(value)* shall expand to a signed integer constant with the specified value
 11923 and type **int_leastN_t**. The macro *UINTN_C(value)* shall expand to an unsigned integer
 11924 constant with the specified value and type **uint_leastN_t**. For example, if **uint_least64_t** is a
 11925 name for the type **unsigned long long**, then *UINT64_C(0x123)* might expand to the integer
 11926 constant $0x123ULL$.

11927 • Macros for greatest-width integer constants

11928 The following macro expands to an integer constant having the value specified by its
 11929 argument and the type **intmax_t**:

11930 **INTMAX_C(value)**

11931 The following macro expands to an integer constant having the value specified by its
 11932 argument and the type **uintmax_t**:

11933 **UINTMAX_C(value)**

11934 **APPLICATION USAGE**

11935 None.

11936 **RATIONALE**

11937 The <stdint.h> header is a subset of the <inttypes.h> header more suitable for use in
11938 freestanding environments, which might not support the formatted I/O functions. In some
11939 environments, if the formatted conversion support is not wanted, using this header instead of
11940 the <inttypes.h> header avoids defining such a large number of macros.

11941 **FUTURE DIRECTIONS**

11942 **typedef** names beginning with **int** or **uint** and ending with **_t** may be added to the types defined
11943 in the <stdint.h> header. Macro names beginning with **INT** or **UINT** and ending with **_MAX**,
11944 **_MIN**, or **_C** may be added to the macros defined in the <stdint.h> header.

11945 **SEE ALSO**

11946 <signal.h>, <stddef.h>, <wchar.h>, <inttypes.h>

11947 **CHANGE HISTORY**

11948 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

11949 **NAME**11950 `stdio.h` — standard buffered input/output11951 **SYNOPSIS**11952 `#include <stdio.h>`11953 **DESCRIPTION**

11954 *CX* The functionality described on this reference page extends the ISO C standard. Applications
11955 shall define the appropriate feature test macro (see the System Interfaces volume of
11956 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
11957 symbols in this header.

11958 The **<stdio.h>** header shall define the following macro names as positive integral constant
11959 expressions:

11960	<code>{BUFSIZ}</code>	Size of <stdio.h> buffers.
11961	<code>{FILENAME_MAX}</code>	Maximum size in bytes of the longest file name string that the 11962 implementation guarantees can be opened.
11963	<code>{FOPEN_MAX}</code>	Number of streams which the implementation guarantees can be open 11964 simultaneously. The value is at least eight.
11965	<code>{_IOFBF}</code>	Input/output fully buffered.
11966	<code>{_IOLBF}</code>	Input/output line buffered.
11967	<code>{_IONBF}</code>	Input/output unbuffered.
11968	<code>{L_ctermid}</code>	Maximum size of character array to hold <i>ctermid()</i> output.
11969	<code>{L_tmpnam}</code>	Maximum size of character array to hold <i>tmpnam()</i> output.
11970	<code>{SEEK_CUR}</code>	Seek relative to current position.
11971	<code>{SEEK_END}</code>	Seek relative to end-of-file.
11972	<code>{SEEK_SET}</code>	Seek relative to start-of-file.
11973	<code>{TMP_MAX}</code>	Minimum number of unique file names generated by <i>tmpnam()</i> . 11974 <i>XSI</i> Maximum number of times an application can call <i>tmpnam()</i> reliably. 11975 The value of <code>{TMP_MAX}</code> is at least 10,000.

11976 The following macro name shall be defined as a negative integral constant expression:

11977 `EOF` End-of-file return value.

11978 The following macro name shall be defined as a null pointer constant:

11979 `NULL` Null pointer.

11980 The following macro name shall be defined as a string constant:

11981 *XSI* `P_tmpdir` Default directory prefix for *tmpnam()*.

11982 The following macro names shall be defined as expressions of type pointer to FILE:

11983 `stderr` Standard error output stream.

11984 `stdin` Standard input stream.

11985 `stdout` Standard output stream.

11986 The following data types shall be defined through **typedef**:

11987	FILE	A structure containing information about a file.
11988	fpos_t	Type containing all information needed to specify uniquely every
11989		position within a file.
11990 XSI	va_list	As described in <stdarg.h>.
11991	size_t	As described in <stddef.h>.
11992	The following shall be declared as functions and may also be defined as macros. Function	
11993	prototypes shall be provided for use with an ISO C standard compiler.	
11994	void	clearerr(FILE *);
11995	char	*ctermid(char *);
11996	int	fclose(FILE *);
11997	FILE	*fdopen(int, const char *);
11998	int	feof(FILE *);
11999	int	ferror(FILE *);
12000	int	fflush(FILE *);
12001	int	fgetc(FILE *);
12002	int	fgetpos(FILE *restrict, fpos_t *restrict);
12003	char	*fgets(char *restrict, int, FILE *restrict);
12004	int	fileno(FILE *);
12005 TSF	void	flockfile(FILE *);
12006	FILE	*fopen(const char *restrict, const char *restrict);
12007	int	fprintf(FILE *restrict, const char *restrict, ...);
12008	int	fputc(int, FILE *);
12009	int	fputs(const char *restrict, FILE *restrict);
12010	size_t	fread(void *restrict, size_t, size_t, FILE *restrict);
12011	FILE	*freopen(const char *restrict, const char *restrict,
12012		FILE *restrict);
12013	int	fscanf(FILE *restrict, const char *restrict, ...);
12014	int	fseek(FILE *, long, int);
12015 XSI	int	fseeko(FILE *, off_t, int);
12016	int	fsetpos(FILE *, const fpos_t *);
12017	long	ftell(FILE *);
12018 XSI	off_t	ftello(FILE *);
12019 TSF	int	ftrylockfile(FILE *);
12020	void	funlockfile(FILE *);
12021	size_t	fwrite(const void *restrict, size_t, size_t, FILE *restrict);
12022	int	getc(FILE *);
12023	int	getchar(void);
12024 TSF	int	getc_unlocked(FILE *);
12025	int	getchar_unlocked(void);
12026	char	*gets(char *);
12027	int	pclose(FILE *);
12028	void	perror(const char *);
12029	FILE	*popen(const char *, const char *);
12030	int	printf(const char *restrict, ...);
12031	int	putc(int, FILE *);
12032	int	putchar(int);
12033 TSF	int	putc_unlocked(int, FILE *);
12034	int	putchar_unlocked(int);
12035	int	puts(const char *);
12036	int	remove(const char *);

```

12037     int        rename(const char *, const char *);
12038     void        rewind(FILE *);
12039     int         scanf(const char *restrict, ...);
12040     void        setbuf(FILE *restrict, char *restrict);
12041     int         setvbuf(FILE *restrict, char *restrict, int, size_t);
12042 XSI     int         snprintf(char *restrict, size_t, const char *restrict, ...);
12043     int         sprintf(char *restrict, const char *restrict, ...);
12044     int         sscanf(const char *restrict, const char *restrict, int ...);
12045 XSI     char        *tempnam(const char *, const char *);
12046     FILE        *tmpfile(void);
12047     char        *tmpnam(char *);
12048     int         ungetc(int, FILE *);
12049     int         vfprintf(FILE *restrict, const char *restrict, va_list);
12050     int         vfscanf(FILE *restrict, const char *restrict, va_list);
12051     int         vprintf(const char *restrict, va_list);
12052     int         vscanf(const char *restrict, va_list);
12053 XSI     int         vsnprintf(char *restrict, size_t, const char *restrict, va_list);
12054     int         vsprintf(char *restrict, const char *restrict, va_list);
12055     int         vsscanf(const char *restrict, const char *restrict, va_list arg);

```

12056 XSI **Inclusion of the <stdio.h> header may also make visible all symbols from <stddef.h>.**

12057 **APPLICATION USAGE**

12058 None.

12059 **RATIONALE**

12060 None.

12061 **FUTURE DIRECTIONS**

12062 None.

12063 **SEE ALSO**

12064 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *clearerr()*, *ctermid()*,
12065 *fclose()*, *fdopen()*, *fgetc()*, *fgetpos()*, *ferror()*, *feof()*, *fflush()*, *fgets()*, *fileno()*, *flockfile()*, *fopen()*,
12066 *fputc()*, *fputs()*, *fread()*, *freopen()*, *fseek()*, *fsetpos()*, *ftell()*, *fwrite()*, *getc()*, *getc_unlocked()*,
12067 *getwchar()*, *getchar()*, *getopt()*, *gets()*, *pclose()*, *perror()*, *popen()*, *printf()*, *putc()*, *putchar()*, *puts()*,
12068 *putwchar()*, *remove()*, *rename()*, *rewind()*, *scanf()*, *setbuf()*, *setvbuf()*, *sscanf()*, *stdin()*, *system()*,
12069 *tempnam()*, *tmpfile()*, *tmpnam()*, *ungetc()*, *vfscanf()*, *vscanf()*, *vprintf()*, *vsscanf()*

12070 **CHANGE HISTORY**

12071 First released in Issue 1. Derived from Issue 1 of the SVID.

12072 **Issue 4**

12073 The constant {L_cuserid} and the external variables *optarg*, *opterr*, *optind*, and *optopt* are marked
12074 as extensions and TO BE WITHDRAWN.

12075 The minimum allowable value of {TMP_MAX}, 10,000 on XSI-conformant systems, has been
12076 marked as an extension.

12077 The P_tmpdir constant is moved from the APPLICATION USAGE section to the DESCRIPTION
12078 and marked as an extension. The remainder of the APPLICATION USAGE section is removed.

12079 References to the *va_list* and *size_t* types are added to the DESCRIPTION.

12080 Function declarations of the *cuserid()*, *getopt()*, and *tempnam()* functions and the *va_list* type are
12081 marked as extensions.

- 12082 The *cuserid()* and *getopt()* functions are marked TO BE WITHDRAWN.
- 12083 A warning is added indicating that inclusion of <stdio.h> may also make visible all symbols
12084 from <stddef.h>.
- 12085 The following changes are incorporated for alignment with the ISO C standard:
- 12086 • The function declarations in this header are expanded to full ISO C standard prototypes.
 - 12087 • The DESCRIPTION is restructured to group lists of macro names according to how they are
12088 defined by an implementation (for example, whether they are integral constant expressions,
12089 pointer constants, or string constants).
 - 12090 • The constant {FILENAME_MAX} is added to the list of integral constant expressions. The
12091 text of {FOPEN_MAX} has also been changed for consistency with the ISO C standard.
 - 12092 • The data type **fpos_t** is moved from the APPLICATION USAGE section to the
12093 DESCRIPTION.
 - 12094 • The *fgetpos()* and *fsetpos()* functions are added to the list of functions declared in this header.
- 12095 **Issue 5**
- 12096 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.
- 12097 Large File System extensions are added.
- 12098 The constant {L_cuserid} and the external variables *optarg*, *opterr*, *optind*, and *optopt* are marked
12099 as extensions and LEGACY.
- 12100 The *cuserid()* and *getopt()* functions are marked LEGACY.
- 12101 **Issue 6**
- 12102 The constant {L_cuserid} and the external variables *optarg*, *opterr*, *optind*, and *optopt* are removed
12103 as they were previously marked LEGACY.
- 12104 The *cuserid()* and *getopt()* functions are removed as they were previously marked LEGACY.
- 12105 Several functions are marked as part of the `_POSIX_THREAD_SAFE_FUNCTIONS` option.
- 12106 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

12107 NAME

12108 stdlib.h — standard library definitions

12109 SYNOPSIS

12110 #include <stdlib.h>

12111 DESCRIPTION

12112 cx The functionality described on this reference page extends the ISO C standard. Applications
12113 shall define the appropriate feature test macro (see the System Interfaces volume of
12114 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
12115 symbols in this header.

12116 The <stdlib.h> header shall define the following macro names:

12117 EXIT_FAILURE Unsuccessful termination for *exit()*; evaluates to a non-zero value.

12118 EXIT_SUCCESS Successful termination for *exit()*; evaluates to 0.

12119 NULL Null pointer.

12120 {RAND_MAX} Maximum value returned by *rand()*; at least 32,767.

12121 {MB_CUR_MAX} Integer expression whose value is the maximum number of bytes in a
12122 character specified by the current locale.

12123 The following data types shall be defined through **typedef**:

12124 **div_t** Structure type returned by the *div()* function.

12125 **ldiv_t** Structure type returned by the *ldiv()* function.

12126 **lldiv_t** Structure type returned by the *lldiv()* function.

12127 **size_t** As described in <stddef.h>.

12128 **wchar_t** As described in <stddef.h>.

12129 In addition, the following symbolic names and macros shall be defined as in <sys/wait.h>, for
12130 use in decoding the return value from *system()*:

12131 xsi WNOHANG

12132 WUNTRACED

12133 WEXITSTATUS

12134 WIFEXITED

12135 WIFSIGNALED

12136 WIFSTOPPED

12137 WSTOPSIG

12138 WTERMSIG

12139

12140 The following shall be declared as functions and may also be defined as macros. Function
12141 prototypes shall be provided for use with an ISO C standard compiler.

12142 void _Exit(int);

12143 xsi long a64l(const char *);

12144 void abort(void);

12145 int abs(int);

12146 int atexit(void (*)(void));

12147 double atof(const char *);

12148 int atoi(const char *);

12149 long atol(const char *);

```

12150     long long    atoll(const char *);
12151     void        *bsearch(const void *, const void *, size_t, size_t,
12152                       int (*)(const void *, const void *));
12153     void        *calloc(size_t, size_t);
12154     div_t       div(int, int);
12155 XSI     double    drand48(void);
12156     char        *ecvt(double, int, int *restrict, int *restrict); (LEGACY)
12157     double     erand48(unsigned short[3]);
12158     void        exit(int);
12159 XSI     char        *fcvt(double, int, int *restrict, int *restrict); (LEGACY)
12160     void        free(void *);
12161 XSI     char        *gcvt(double, int, char *); (LEGACY)
12162     char        *getenv(const char *);
12163 XSI     int        getsubopt(char **, char *const *, char **);
12164     int        grantpt(int);
12165     char        *initstate(unsigned, char *, size_t);
12166     long        jrand48(unsigned short[3]);
12167     char        *l64a(long);
12168     long        labs(long);
12169 XSI     void        lcong48(unsigned short[7]);
12170     ldiv_t     ldiv(long, long);
12171     long long  llabs(long long);
12172 XSI     long        lrand48(void);
12173     void        *malloc(size_t);
12174     int        mblen(const char *, size_t);
12175     size_t     mbstowcs(wchar_t *restrict, const char *restrict, size_t);
12176     int        mbtowc(wchar_t *restrict, const char *restrict, size_t);
12177 XSI     char        *mktemp(char *); (LEGACY)
12178     int        mkstemp(char *);
12179     long        mrand48(void);
12180     long        nrand48(unsigned short[3]);
12181 ADV     int        posix_memalign(void **, size_t, size_t);
12182 XSI     char        *ptsname(int);
12183     int        putenv(char *);
12184     void        qsort(void *, size_t, size_t, int (*)(const void *,
12185                       const void *));
12186     int        rand(void);
12187 TSF     int        rand_r(unsigned *);
12188 XSI     long        random(void);
12189     void        *realloc(void *, size_t);
12190 XSI     char        *realpath(const char *restrict, char *restrict);
12191     unsigned short seed48(unsigned short[3]);
12192     int        setenv(const char *, const char *, int);
12193     void        setkey(const char *);
12194     char        *setstate(const char *);
12195     void        srand(unsigned);
12196 XSI     void        srand48(long);
12197     void        srandom(unsigned);
12198     double    strtod(const char *restrict, char **restrict);
12199     float     strtod(const char *restrict, char **restrict);
12200     long        strtol(const char *restrict, char **restrict, int);
12201     long double strtold(const char *restrict, char **restrict);

```

```

12202     long long      strtoll(const char *restrict, char **restrict, int);
12203     unsigned long strtoul(const char *restrict, char **restrict, int);
12204     long long      strtoull(const char *restrict, char **restrict, int);
12205     int            system(const char *);
12206 XSI    int            unlockpt(int);
12207     int            unsetenv(const char *);
12208     size_t        wcstombs(char *restrict, const wchar_t *restrict, size_t);
12209     int            wctomb(char *, wchar_t);

```

12210 XSI Inclusion of the **<stdlib.h>** header may also make visible all symbols from **<stddef.h>**,
12211 **<limits.h>**, **<math.h>**, and **<sys/wait.h>**.

12212 APPLICATION USAGE

12213 None.

12214 RATIONALE

12215 None.

12216 FUTURE DIRECTIONS

12217 None.

12218 SEE ALSO

12219 **<sys/types.h>**, the System Interfaces volume of IEEE Std. 1003.1-200x, *_Exit()*, *a64l()*, *abort()*,
12220 *abs()*, *atexit()*, *atof()*, *atoi()*, *atol()*, *atoll()*, *bsearch()*, *calloc()*, *div()*, *drand48()*, *erand48()*, *exit()*,
12221 *free()*, *getenv()*, *getsubopt()*, *grantpt()*, *initstate()*, *jrand48()*, *l64a()*, *labs()*, *lcong48()*, *ldiv()*, *llabs()*,
12222 *lldiv()*, *lrand48()*, *malloc()*, *mblen()*, *mbstowcs()*, *mbtowc()*, *mkstemp()*, *mrand48()*, *nrand48()*,
12223 *posix_memalign()*, *ptsname()*, *putenv()*, *qsort()*, *rand()*, *realloc()*, *realpath()*, *setstate()*, *srand()*,
12224 *srand48()*, *srandom()*, *strtod()*, *strtof()*, *strtol()*, *strtold()*, *strtoll()*, *strtoul()*, *strtoull()*, *unlockpt()*,
12225 *wctombs()*, *wctomb()*

12226 CHANGE HISTORY

12227 First released in Issue 3.

12228 Issue 4

12229 A reference is added to **<stddef.h>** and **<wchar.h>** for the definition of **size_t**.

12230 A reference is added to **<sys/wait.h>** for definitions of the symbolic names and macros defined
12231 for decoding the return value from the *system()* function. This reference and the symbolic names
12232 and macros are marked as an extension.

12233 The names *drand48()*, *erand48()*, *jrand48()*, *lcong48()*, *lrand48()*, *mrand48()*, *nrand48()*, *putenv()*,
12234 *seed48()*, *setkey()*, and *srand48()* are added to the list of functions declared in this header and
12235 marked as extensions.

12236 A warning is added indicating that inclusion of **<stdlib.h>** may also make visible all symbols
12237 from **<stddef.h>**, **<limits.h>**, **<math.h>**, and **<sys/wait.h>**.

12238 The APPLICATION USAGE section is removed.

12239 The following changes are incorporated for alignment with the ISO C standard:

- 12240 • The function declarations in this header are expanded to full ISO C standard prototypes.
- 12241 • The maximum value of {RAND_MAX} is defined.
- 12242 • The name {MB_CUR_MAX} is added to the list of macro names defined in this header, while
12243 **div_t** and **ldiv_t** are added to the list of defined types.
- 12244 • The names *atexit()*, *div()*, *labs()*, *ldiv()*, *mblen()*, *mbstowcs()*, *mbtowc()*, *strtoul()*, *wctombs()*,
12245 and *wctomb()* are added to the list of functions declared in this header.

12246 **Issue 4, Version 2**

12247 For X/OPEN UNIX conformance, the *a64l()*, *ecvt()*, *fcvt()*, *gcvt()*, *getsubopt()*, *grantpt()*,
12248 *initstate()*, *l64a()*, *mktemp()*, *mkstemp()*, *ptsname()*, *random()*, *realpath()*, *setstate()*, *srandom()*,
12249 *ttyslot()*, *unlockpt()*, and *valloc()* functions are added to the list of functions declared in this
12250 header.

12251 **Issue 5**

12252 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

12253 The *ttyslot()* and *valloc()* functions are marked LEGACY.

12254 The type of the third argument to *initstate()* is changed from **int** to **size_t**. The type of the return
12255 value from *setstate()* is changed from **char** to **char***, and the type of the first argument is changed
12256 from **char*** to **const char***.

12257 **Issue 6**

12258 The Open Group corrigenda item U021/1 has been applied, correcting the prototype for
12259 *realpath()* to be consistent with the reference page.

12260 The Open Group corrigenda item U028/13 has been applied, correcting the prototype for
12261 *putenv()* to be consistent with the reference page.

12262 The *rand_r()* function is marked as part of the `_POSIX_THREAD_SAFE_FUNCTIONS` option.

12263 Function prototypes for *setenv()* and *unsetenv()* are added.

12264 The *posix_memalign()* function is added for alignment with IEEE Std. 1003.1d-1999. |

12265 This reference page is updated to align with the ISO/IEC 9899:1999 standard. |

12266 The *ecvt()*, *fcvt()*, *gcvt()*, and *mktemp()* functions are marked LEGACY. |

12267 **NAME**12268 `string.h` — string operations12269 **SYNOPSIS**12270 `#include <string.h>`12271 **DESCRIPTION**

12272 CX The functionality described on this reference page extends the ISO C standard. Applications
 12273 shall define the appropriate feature test macro (see the System Interfaces volume of
 12274 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 12275 symbols in this header.

12276 The **<string.h>** header shall define the following:12277 **NULL** Null pointer constant.12278 **size_t** As described in **<stddef.h>**.

12279 The following shall be declared as functions and may also be defined as macros. Function
 12280 prototypes shall be provided for use with an ISO C standard compiler.

12281 XSI `void *memcpy(void *restrict, const void *restrict, int, size_t);`12282 `void *memchr(const void *, int, size_t);`12283 `int memcmp(const void *, const void *, size_t);`12284 `void *memcpy(void *restrict, const void *restrict, size_t);`12285 `void *memmove(void *, const void *, size_t);`12286 `void *memset(void *, int, size_t);`12287 `char *strcat(char *restrict, const char *restrict);`12288 `char *strchr(const char *, int);`12289 `int strcmp(const char *, const char *);`12290 `int strcoll(const char *, const char *);`12291 `char *strcpy(char *restrict, const char *restrict);`12292 `size_t strcspn(const char *, const char *);`12293 XSI `char *strdup(const char *);`12294 `char *strerror(int);`12295 `size_t strlen(const char *);`12296 `char *strncat(char *restrict, const char *restrict, size_t);`12297 `int strncmp(const char *, const char *, size_t);`12298 `char *strncpy(char *restrict, const char *restrict, size_t);`12299 `char *strpbrk(const char *, const char *);`12300 `char *strrchr(const char *, int);`12301 `size_t strspn(const char *, const char *);`12302 `char *strstr(const char *, const char *);`12303 `char *strtok(char *restrict, const char *restrict);`12304 TSF `char *strtok_r(char *, const char *, char **);`12305 `size_t strxfrm(char *restrict, const char *restrict, size_t);`12306 XSI Inclusion of the **<string.h>** header may also make visible all symbols from **<stddef.h>**.

12307 **APPLICATION USAGE**

12308 None.

12309 **RATIONALE**

12310 None.

12311 **FUTURE DIRECTIONS**

12312 None.

12313 **SEE ALSO**

12314 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *memccpy()*, *memchr()*,
 12315 *memcmp()*, *memcpy()*, *memmove()*, *memset()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*, *strcpy()*,
 12316 *strcspn()*, *strdup()*, *strerror()*, *strlen()*, *strncat()*, *strncmp()*, *strncpy()*, *strpbrk()*, *strrchr()*, *strspn()*,
 12317 *strstr()*, *strtok()*, *strxfrm()*

12318 **CHANGE HISTORY**

12319 First released in Issue 1. Derived from Issue 1 of the SVID.

12320 **Issue 4**12321 A reference is added to <stddef.h> for the definition of **size_t**.12322 The *memccpy()* function is marked as an extension.

12323 A warning is added indicating that inclusion of <string.h> may also make visible all symbols
 12324 from <stddef.h>.

12325 The APPLICATION USAGE section is removed.

12326 The following changes are incorporated for alignment with the ISO C standard:

- 12327 • The function declarations in this header are expanded to full ISO C standard prototypes.
- 12328 • The name *memmove()* is added to the list of functions declared in this header.

12329 **Issue 4, Version 2**

12330 For X/OPEN UNIX conformance, the *strdup()* function is added to the list of functions declared
 12331 in this header.

12332 **Issue 5**

12333 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

12334 **Issue 6**12335 The *strtok_r()* function is marked as part of the `_POSIX_THREAD_SAFE_FUNCTIONS` option.

12336 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

12337 **NAME**

12338 strings.h — string operations

12339 **SYNOPSIS**

12340 XSI #include <strings.h>

12341

12342 **DESCRIPTION**

12343 The following shall be declared as functions and may also be defined as macros. Function
12344 prototypes shall be provided for use with an ISO C standard compiler.

12345 int bcmp(const void *, const void *, size_t); (**LEGACY**)

12346 void bcopy(const void *, void *, size_t); (**LEGACY**)

12347 void bzero(void *, size_t); (**LEGACY**)

12348 int ffs(int);

12349 char *index(const char *, int); (**LEGACY**)

12350 char *rindex(const char *, int); (**LEGACY**)

12351 int strcasecmp(const char *, const char *);

12352 int strncasecmp(const char *, const char *, size_t);

12353 The `size_t` type shall be defined through `typedef` as described in <stddef.h>.

12354 **APPLICATION USAGE**

12355 None.

12356 **RATIONALE**

12357 None.

12358 **FUTURE DIRECTIONS**

12359 None.

12360 **SEE ALSO**

12361 <stddef.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, `ffs()`, `strcasecmp()`,

12362 `strncasecmp()`

12363 **CHANGE HISTORY**

12364 First released in Issue 4, Version 2.

12365 **Issue 6**

12366 The Open Group corrigenda item U021/2 has been applied, correcting the prototype for `index()`
12367 to be consistent with the reference page.

12368 The `bcmp()`, `bcopy()`, `bzero()`, `index()`, and `rindex()` functions are marked LEGACY.

12369 **NAME**

12370 `stropts.h` — STREAMS interface (**STREAMS**)

12371 **SYNOPSIS**

12372 XSR `#include <stropts.h>`

12373

12374 **DESCRIPTION**

12375 The `<stropts.h>` header shall define the **bandinfo** structure that includes at least the following
 12376 members:

12377 `unsigned char bi_pri`
 12378 `int bi_flag`

12379 The `<stropts.h>` header shall define the **strpeek** structure that includes at least the following
 12380 members:

12381 `struct strbuf ctlbuf`
 12382 `struct strbuf databuf`
 12383 `t_uscalar_t flags`

12384 The `<stropts.h>` header shall define the **strbuf** structure that includes at least the following
 12385 members:

12386 `int maxlen` Maximum buffer length.
 12387 `int len` Length of data.
 12388 `char *buf` Pointer to buffer.

12389 The `<stropts.h>` header shall define the **strfdinsert** structure that includes at least the following
 12390 members:

12391 `struct strbuf ctlbuf`
 12392 `struct strbuf databuf`
 12393 `t_uscalar_t flags`
 12394 `int fildes`
 12395 `int offset`

12396 The `<stropts.h>` header shall define the **striocctl** structure that includes at least the following
 12397 members:

12398 `int ic_cmd`
 12399 `int ic_timeout`
 12400 `int ic_len`
 12401 `char *ic_dp`

12402 The `<stropts.h>` header shall define the **strrecvfd** structure that includes at least the following
 12403 members:

12404 `int fd`
 12405 `uid_t uid`
 12406 `gid_t gid`

12407 The **uid_t** and **gid_t** types shall be defined through **typedef** as described in `<sys/types.h>`.

12408 The **t_uscalar_t** type shall be defined as described in the referenced XNS, Issue 5 specification,
 12409 `<xti.h>`.

12410 The `<stropts.h>` header shall define the **str_list** structure that includes at least the following
 12411 members:

```

12412     int                sl_nmods
12413     struct str_mlist  *sl_modlist

```

12414 The <stropts.h> header shall define the **str_mlist** structure that includes at least the following
 12415 member:

```

12416     char                l_name[ FMNAMESZ+1 ]

```

12417 At least the following macros shall be defined for use as the *request* argument to *ioctl()*:

12418 **I_PUSH** Push STREAMS module onto the top of the current STREAM, just below the
 12419 STREAM head.

12420 **I_POP** Remove STREAMS module from just below the STREAM head.

12421 **I_LOOK** Retrieve the name of the module just below the STREAM head and place it in
 12422 a character string. At least the following macros shall be defined for use as the
 12423 *arg* argument:

12424 **FMNAMESZ** The minimum size in bytes of the buffer referred to by the
 12425 *arg* argument.

12426 **I_FLUSH** This request flushes all input and/or output queues, depending on the value
 12427 of the *arg* argument. At least the following macros shall be defined for use as
 12428 the *arg* argument:

12429 **FLUSHR** Flush read queues.

12430 **FLUSHW** Flush write queues.

12431 **FLUSHRW** Flush read and write queues.

12432 **I_FLUSHBAND** Flush only band specified.

12433 **I_SETSIG** Informs the STREAM head that the process wants the SIGPOLL signal issued
 12434 (see *signal()*) when a particular event has occurred on the STREAM.

12435 The <stropts.h> header shall define the following possible values for *arg* when
 12436 **I_SETSIG** is specified:

12437 **S_RDNORM** A normal (priority band set to 0) message has arrived at the
 12438 head of a STREAM head read queue.

12439 **S_RDBAND** A message with a non-zero priority band has arrived at the
 12440 head of a STREAM head read queue.

12441 **S_INPUT** A message, other than a high-priority message, has arrived
 12442 at the head of a STREAM head read queue.

12443 **S_HIPRI** A high-priority message is present on a STREAM head read
 12444 queue.

12445 **S_OUTPUT** The write queue for normal data (priority band 0) just
 12446 below the STREAM head is no longer full. This notifies the
 12447 process that there is room on the queue for sending (or
 12448 writing) normal data downstream.

12449 **S_WRNORM** Same as **S_OUTPUT**.

12450 **S_WRBAND** The write queue for a non-zero priority band just below the
 12451 STREAM head is no longer full.

12452		S_MSG	A STREAMS signal message that contains the SIGPOLL signal reaches the front of the STREAM head read queue.
12453			
12454		S_ERROR	Notification of an error condition reaches the STREAM head.
12455			
12456		S_HANGUP	Notification of a hangup reaches the STREAM head.
12457		S_BANDURG	When used in conjunction with S_RDBAND, SIGURG is generated instead of SIGPOLL when a priority message reaches the front of the STREAM head read queue.
12458			
12459			
12460	I_GETSIG		Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal.
12461			
12462	I_FIND		Compares the names of all modules currently present in the STREAM to the name pointed to by <i>arg</i> .
12463			
12464	I_PEEK		Allows a process to retrieve the information in the first message on the STREAM head read queue without taking the message off the queue. At least the following macros are defined for use as the <i>arg</i> argument:
12465			
12466			
12467		RS_HIPRI	Only look for high-priority messages.
12468	I_SRDOPT		Sets the read mode. At least the following macros shall be defined for use as the <i>arg</i> argument:
12469			
12470		RNORM	Byte-STREAM mode, the default.
12471		RMSGD	Message-discard mode.
12472		RMSGN	Message-nondiscard mode.
12473		RPROTNORM	Fail <i>read()</i> with [EBADMSG] if a message containing a control part is at the front of the STREAM head read queue.
12474			
12475		RPROTDAT	Deliver the control part of a message as data when a process issues a <i>read()</i> .
12476			
12477		RPROTDIS	Discard the control part of a message, delivering any data part, when a process issues a <i>read()</i> .
12478			
12479	I_GRDOPT		Returns the current read mode setting.
12480	I_NREAD		Counts the number of data bytes in data blocks in the first message on the STREAM head read queue.
12481			
12482	I_FDINSERT		Creates a message from the specified buffer(s), adds information about another STREAM, and sends the message downstream.
12483			
12484	I_STR		Constructs an internal STREAMS <i>ioctl()</i> message and sends that message downstream.
12485			
12486	I_SWROPT		Sets the write mode. At least the following macros are defined for use as the <i>arg</i> argument:
12487			
12488		SNDZERO	Send a zero-length message downstream when a <i>write()</i> of 0 bytes occurs.
12489			
12490	I_GWROPT		Returns the current write mode setting.
12491	I_SENDFD		Requests the STREAM associated with <i>fildev</i> to send a message, containing a file pointer, to the STREAM head at the other end of a STREAMS pipe.
12492			

12493	I_RECVFD	Retrieves the file descriptor associated with the message sent by an
12494		I_SENDFD <i>ioctl()</i> over a STREAMS pipe.
12495	I_LIST	This request allows the process to list all the module names on the STREAM,
12496		up to and including the topmost driver name.
12497	I_ATMARK	This request allows the process to see if the current message on the STREAM
12498		head read queue is “marked” by some module downstream. At least the
12499		following macros are defined for use as the <i>arg</i> argument:
12500	ANYMARK	Check if the message is marked.
12501	LASTMARK	Check if the message is the last one marked on the queue.
12502	I_CKBAND	Check if the message of a given priority band exists on the STREAM head
12503		read queue.
12504	I_GETBAND	Return the priority band of the first message on the STREAM head read
12505		queue.
12506	I_CANPUT	Check if a certain band is writable.
12507	I_SETCLTIME	Allow the process to set the time the STREAM head delays when a STREAM
12508		is closing and there is data on the write queues.
12509	I_GETCLTIME	Returns the close time delay.
12510	I_LINK	Connects two STREAMS.
12511	I_UNLINK	Disconnects the two STREAMS. The header shall define at least the following
12512		value for <i>arg</i> :
12513	MUXID_ALL	Unlink all STREAMS linked to the STREAM associated with
12514		<i>files</i> .
12515	I_PLINK	Connects two STREAMS with a persistent link.
12516	I_PUNLINK	Disconnects the two STREAMS that were connected with a persistent link.
12517		The following macros shall be defined for <i>getmsg()</i> , <i>getpmsg()</i> , <i>putmsg()</i> , and <i>putpmsg()</i> :
12518	MSG_ANY	Receive any message.
12519	MSG_BAND	Receive message from specified band.
12520	MSG_HIPRI	Send/receive high-priority message.
12521	MORECTL	More control information is left in message.
12522	MOREDATA	More data is left in message.
12523		The <stropts.h> header may make visible all of the symbols from <unistd.h> .
12524		The following shall be declared as functions in the <stropts.h> header and may also be defined
12525		as macros. Function prototypes shall be provided for use with an ISO C standard compiler.
12526	int	isastream(int);
12527	int	getmsg(int, struct strbuf *restrict, struct strbuf *restrict,
12528		int *restrict);
12529	int	getpmsg(int, struct strbuf *restrict, struct strbuf *restrict,
12530		int *restrict, int *restrict);
12531	int	ioctl(int, int, ...);
12532	int	putmsg(int, const struct strbuf *, const struct strbuf *, int);
12533	int	putpmsg(int, const struct strbuf *, const struct strbuf *, int,

```
12534         int);
12535     int    fattach(int, const char *);
12536     int    fdetach(const char *);
```

12537 APPLICATION USAGE

12538 None.

12539 RATIONALE

12540 None.

12541 FUTURE DIRECTIONS

12542 None.

12543 SEE ALSO

12544 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *close()*, *fcntl()*, *getmsg()*,
12545 *ioctl()*, *open()*, *pipe()*, *read()*, *poll()*, *putmsg()*, *signal()*, *write()* the XNS, Issue 5 specification,
12546 <xti.h>

12547 CHANGE HISTORY

12548 First released in Issue 4, Version 2.

12549 Issue 5

12550 The *flags* member of the **strpeek** and **strfdinsert** structures are changed from **type long** to
12551 **t_uscalar_t**.

12552 Issue 6

12553 This header is marked as part of the XSI STREAMS Option Group.

12554 The **restrict** keyword is added to the prototypes for *getmsg()* and *getpmsg()*.

12555 **NAME**

12556 sys/ipc.h — XSI interprocess communication access structure

12557 **SYNOPSIS**

12558 XSI #include <sys/ipc.h>

12559

12560 **DESCRIPTION**

12561 The **<sys/ipc.h>** header is used by three mechanisms for XSI interprocess communication (IPC):
12562 messages, semaphores, and shared memory. All use a common structure type, **ipc_perm** to pass
12563 information used in determining permission to perform an IPC operation.

12564 The **ipc_perm** structure shall contain the following members:

12565	uid_t	uid	Owner's user ID.
12566	gid_t	gid	Owner's group ID.
12567	uid_t	cuid	Creator's user ID.
12568	gid_t	cgid	Creator's group ID.
12569	mode_t	mode	Read/write permission.

12570 The **uid_t**, **gid_t**, **mode_t**, and **key_t** types shall be defined as described in **<sys/types.h>**.

12571 Definitions shall be provided for the following constants:

12572 Mode bits:

12573 **IPC_CREAT** Create entry if key does not exist.

12574 **IPC_EXCL** Fail if key exists.

12575 **IPC_NOWAIT** Error if request must wait.

12576 Keys:

12577 **IPC_PRIVATE** Private key.

12578 Control commands:

12579 **IPC_RMID** Remove identifier.

12580 **IPC_SET** Set options.

12581 **IPC_STAT** Get options.

12582 The following shall be declared as a function and may also be defined as a macro. Function
12583 prototypes shall be provided for use with an ISO C standard compiler.

12584 `key_t ftok(const char *, int);`

12585 **APPLICATION USAGE**

12586 None.

12587 **RATIONALE**

12588 None.

12589 **FUTURE DIRECTIONS**

12590 None.

12591 **SEE ALSO**

12592 **<sys/types.h>**, the System Interfaces volume of IEEE Std. 1003.1-200x, *ftok()*

12593 **CHANGE HISTORY**

12594 First released in Issue 2. Derived from System V Release 2.0.

12595 **Issue 4**

12596 The DESCRIPTION is corrected to say that the header “is used by three mechanisms ...”.

12597 Reference to the <sys/types.h> header is added for the definitions of **uid_t**, **gid_t**, and **mode_t**.

12598 **Issue 4, Version 2**

12599 For X/OPEN UNIX conformance, the *ftok()* function is added to the list of functions declared in
12600 this header.

12601 **NAME**12602 `sys/mman.h` — memory management declarations12603 **SYNOPSIS**12604 `#include <sys/mman.h>`12605 **DESCRIPTION**12606 The **<sys/mman.h>** header shall be supported if the implementation supports at least one of the
12607 following options:

- 12608 MF • The Memory Mapped Files option
- 12609 SHM • The Shared Memory Objects option
- 12610 ML • The Process Memory Locking option
- 12611 MPR • The Memory Protection option
- 12612 TYM • The Typed Memory Objects option
- 12613 SIO • The Synchronized Input and Output option
- 12614 ADV • The Advisory Information option
- 12615 TYM • The Typed Memory Objects option

12616 The following protection options shall be defined:

- 12617 code2 **PROT_READ** Page can be read.
- 12618 code2 **PROT_WRITE** Page can be written.
- 12619 code2 **PROT_EXEC** Page can be executed.
- 12620 code2 **PROT_NONE** Page cannot be accessed.

12621 The following *flag* options shall be defined:

- 12622 MF|SHM **MAP_SHARED** Share changes.
- 12623 MF|SHM **MAP_PRIVATE** Changes are private.
- 12624 MF|SHM **MAP_FIXED** Interpret *addr* exactly.

12625 The following flags shall be defined for *msync()*:

- 12626 MF|SIO **MS_ASYNC** Perform asynchronous writes.
- 12627 MF|SIO **MS_SYNC** Perform synchronous writes.
- 12628 MF|SIO **MS_INVALIDATE** Invalidate mappings.

12629 ML The following symbolic constants shall be defined for the *mlockall()* function:

- 12630 ML **MCL_CURRENT** Lock currently mapped pages.
- 12631 ML **MCL_FUTURE** Lock pages that become mapped.

12632 MF|SHM The symbolic constant **MAP_FAILED** shall be defined to indicate a failure from the *mmap()*
12633 function.12634 code1 Values for *advice* used by *posix_madvise()* are as follows:12635 **POSIX_MADV_NORMAL**12636 The application has no advice to give on its behavior with respect to the specified range. It
12637 is the default characteristic if no advice is given for a range of memory.

12638		POSIX_MADV_SEQUENTIAL
12639		The application expects to access the specified range sequentially from lower addresses to higher addresses.
12640		
12641		POSIX_MADV_RANDOM
12642		The application expects to access the specified range in a random order.
12643		POSIX_MADV_WILLNEED
12644		The application expects to access the specified range in the near future.
12645		POSIX_MADV_DONTNEED
12646		The application expects that it will not access the specified range in the near future.
12647		
12648	TYM	The following flags shall be defined for <i>posix_typed_mem_open()</i> :
12649		POSIX_TYPED_MEM_ALLOCATE
12650		Allocate on <i>mmap()</i> .
12651		POSIX_TYPED_MEM_ALLOCATE_CONTIG
12652		Allocate contiguously on <i>mmap()</i> .
12653		POSIX_TYPED_MEM_MAP_ALLOCATABLE Map on <i>mmap()</i> , without affecting allocatability.
12654		
12655		The mode_t , off_t , and size_t types shall be defined as described in <sys/types.h>.
12656	TYM	The <sys/mman.h> header shall define the structure posix_typed_mem_info , which includes at least the following member:
12657		
12658		<code>size_t posix_tmi_length</code> Maximum length which may be allocated
12659		from a typed memory object.
12660		
12661		The following shall be declared in <sys/mman.h> as functions and may also be defined as macros. Function prototypes shall be provided for use with an ISO C standard compiler.
12662		
12663	ML	<code>int mlock(const void *, size_t);</code>
12664		<code>int mlockall(int);</code>
12665	MF SHM	<code>void *mmap(void *, size_t, int, int, int, off_t);</code>
12666	MPR	<code>int mprotect(void *, size_t, int);</code>
12667	MF SIO	<code>int msync(void *, size_t, int);</code>
12668	ML	<code>int munlock(const void *, size_t);</code>
12669		<code>int munlockall(void);</code>
12670	MF SHM	<code>int munmap(void *, size_t);</code>
12671	ADV	<code>int posix_madvise(void *, size_t, int);</code>
12672	TYM	<code>int posix_mem_offset(const void *restrict, size_t, off_t *restrict,</code>
12673		<code>size_t *restrict, int *restrict);</code>
12674		<code>int posix_typed_mem_get_info(int, struct posix_typed_mem_info *);</code>
12675		<code>int posix_typed_mem_open(const char *, int, int);</code>
12676	SHM	<code>int shm_open(const char *, int, mode_t);</code>
12677		<code>int shm_unlink(const char *);</code>
12678		

12679 **APPLICATION USAGE**

12680 None.

12681 **RATIONALE**

12682 None.

12683 **FUTURE DIRECTIONS**

12684 None.

12685 **SEE ALSO**

12686 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *mlock()*, *mlockall()*,
12687 *mmap()*, *mprotect()*, *msync()*, *munlock()*, *munlockall()*, *munmap()*, *posix_mem_offset()*,
12688 *posix_typed_mem_get_info()*, *posix_typed_mem_open()*, *shm_open()*, *shm_unlink()*

12689 **CHANGE HISTORY**

12690 First released in Issue 4, Version 2.

12691 **Issue 5**

12692 Updated for alignment with the POSIX Realtime Extension.

12693 **Issue 6**

12694 The <sys/mman.h> header is marked as dependent on support for either the
12695 _POSIX_MAPPED_FILES, _POSIX_MEMLOCK, or _POSIX_SHARED_MEMORY options.

12696 The following changes are made for alignment with IEEE Std. 1003.1j-2000:

- 12697 • The TYM margin code is added to the list of margin codes for the <sys/mman.h> header line,
12698 as well as for other lines.
- 12699 • The POSIX_TYPED_MEM_ALLOCATE, POSIX_TYPED_MEM_ALLOCATE_CONTIG, and
12700 POSIX_TYPED_MEM_MAP_ALLOCATABLE flags are added.
- 12701 • The **posix_tmi_length** structure is added.
- 12702 • The *posix_mem_offset()*, *posix_typed_mem_get_info()*, and *posix_typed_mem_open()* functions
12703 are added.

12704 The **restrict** keyword is added to the prototype for *posix_mem_offset()*.

12705 IEEE PASC Interpretation 1003.1 #102 is applied adding the prototype for *posix_madvise()*.

12706 **NAME**

12707 sys/msg.h — XSI message queue structures

12708 **SYNOPSIS**

12709 XSI `#include <sys/msg.h>`

12710

12711 **DESCRIPTION**

12712 The <sys/msg.h> header shall define the following constant and members of the structure
12713 **msqid_ds**.

12714 The following data types shall be defined through **typedef**:

12715 **msgqnum_t** Used for the number of messages in the message queue.

12716 **msglen_t** Used for the number of bytes allowed in a message queue.

12717 These types shall be unsigned integer types that are able to store values at least as large as a type
12718 **unsigned short**.

12719 Message operation flag:

12720 **MSG_NOERROR** No error if big message.

12721 The **msqid_ds** structure shall contain the following members:

12722	<code>struct ipc_perm</code>	<code>msg_perm</code>	Operation permission structure.
12723	<code>msgqnum_t</code>	<code>msg_qnum</code>	Number of messages currently on queue.
12724	<code>msglen_t</code>	<code>msg_qbytes</code>	Maximum number of bytes allowed on queue.
12725	<code>pid_t</code>	<code>msg_lspid</code>	Process ID of last <i>msgsnd()</i> .
12726	<code>pid_t</code>	<code>msg_lrpid</code>	Process ID of last <i>msgrcv()</i> .
12727	<code>time_t</code>	<code>msg_stime</code>	Time of last <i>msgsnd()</i> .
12728	<code>time_t</code>	<code>msg_rtime</code>	Time of last <i>msgrcv()</i> .
12729	<code>time_t</code>	<code>msg_ctime</code>	Time of last change.

12730 The **pid_t**, **time_t**, **key_t**, **size_t**, and **ssize_t** types shall be defined as described in <sys/types.h>.

12731 The following shall be declared as functions and may also be defined as macros. Function
12732 prototypes shall be provided for use with an ISO C standard compiler.

12733	<code>int</code>	<code>msgctl(int, int, struct msqid_ds *)</code> ;
12734	<code>int</code>	<code>msgget(key_t, int)</code> ;
12735	<code>ssize_t</code>	<code>msgrcv(int, void *, size_t, long, int)</code> ;
12736	<code>int</code>	<code>msgsnd(int, const void *, size_t, int)</code> ;

12737 In addition, all of the symbols from <sys/ipc.h> shall be defined when this header is included.

12738 **APPLICATION USAGE**

12739 None.

12740 **RATIONALE**

12741 None.

12742 **FUTURE DIRECTIONS**

12743 None.

12744 **SEE ALSO**

12745 <sys/types.h>, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

12746 **CHANGE HISTORY**

12747 First released in Issue 2. Derived from System V Release 2.0.

12748 **Issue 4**

12749 The function declarations in this header are expanded to full ISO C standard prototypes.

12750 Reference to the <sys/types.h> header is added for the definitions of **pid_t**, **time_t**, **key_t**, and
12751 **size_t**.

12752 A statement is added indicating that all symbols in <sys/ipc.h> are defined when this header is
12753 included.

12754 **NAME**

12755 sys/resource.h — definitions for XSI resource operations

12756 **SYNOPSIS**

12757 xsi #include <sys/resource.h>

12758

12759 **DESCRIPTION**

12760 The <sys/resource.h> header shall define the following symbolic constants as possible values of
 12761 the *which* argument of *getpriority()* and *setpriority()*:

12762 PRIO_PROCESS Identifies the *who* argument as a process ID.

12763 PRIO_PGRP Identifies the *who* argument as a process group ID.

12764 PRIO_USER Identifies the *who* argument as a user ID.

12765 The following type shall be defined through **typedef**:

12766 **rlim_t** Unsigned integer type used for limit values.

12767 The following symbolic constants shall be defined:

12768 RLIM_INFINITY A value of **rlim_t** indicating no limit.

12769 RLIM_SAVED_MAX A value of type **rlim_t** indicating an unrepresentable saved hard
 12770 limit.

12771 RLIM_SAVED_CUR A value of type **rlim_t** indicating an unrepresentable saved soft limit.

12772 On implementations where all resource limits are representable in an object of type **rlim_t**,
 12773 RLIM_SAVED_MAX and RLIM_SAVED_CUR need not be distinct from RLIM_INFINITY.

12774 The following symbolic constants shall be defined as possible values of the *who* parameter of
 12775 *getrusage()*:

12776 RUSAGE_SELF Returns information about the current process.

12777 RUSAGE_CHILDREN Returns information about children of the current process.

12778 The <sys/resource.h> header shall define the **rlimit** structure that includes at least the following
 12779 members:

12780 rlim_t rlim_cur The current (soft) limit.

12781 rlim_t rlim_max The hard limit.

12782 The <sys/resource.h> header shall define the **rusage** structure that includes at least the following
 12783 members:

12784 struct timeval ru_utime User time used.

12785 struct timeval ru_stime System time used.

12786 The **timeval** structure shall be defined as described in <sys/time.h>.

12787 The following symbolic constants shall be defined as possible values for the *resource* argument of
 12788 *getrlimit()* and *setrlimit()*:

12789 RLIMIT_CORE Limit on size of core dump file.

12790 RLIMIT_CPU Limit on CPU time per process.

12791 RLIMIT_DATA Limit on data segment size.

12792 RLIMIT_FSIZE Limit on file size.

12793 RLIMIT_NOFILE Limit on number of open files.

12794 RLIMIT_STACK Limit on stack size.

12795 RLIMIT_AS Limit on address space size.

12796 The following are declared as functions and may also be defined as macros. Function prototypes shall be provided for use with an ISO C standard compiler.

12797

12798 int getpriority(int, id_t);

12799 int getrlimit(int, struct rlimit *);

12800 int getrusage(int, struct rusage *);

12801 int setpriority(int, id_t, int);

12802 int setrlimit(int, const struct rlimit *);

12803 The **id_t** type shall be defined through **typedef** as described in **<sys/types.h>**.

12804 Inclusion of the **<sys/resource.h>** header may also make visible all symbols from **<sys/time.h>**.

12805 **APPLICATION USAGE**

12806 None.

12807 **RATIONALE**

12808 None.

12809 **FUTURE DIRECTIONS**

12810 None.

12811 **SEE ALSO**

12812 **<sys/time.h>**, **<sys/types.h>**, the System Interfaces volume of IEEE Std. 1003.1-200x,

12813 *getpriority()*, *getrusage()*, *getrlimit()*

12814 **CHANGE HISTORY**

12815 First released in Issue 4, Version 2.

12816 **Issue 5**

12817 Large File System extensions are added.

12818 **NAME**

12819 sys/select.h — select types

12820 **SYNOPSIS**

12821 #include <sys/select.h>

12822 **DESCRIPTION**12823 The <sys/select.h> header shall define the **timeval** structure that includes at least the following
12824 members:

12825 time_t tv_sec Seconds.

12826 suseconds_t tv_usec Microseconds.

12827 The **time_t** and **suseconds_t** types shall be defined as described in <sys/types.h>.12828 The **sigset_t** type shall be defined as described in <signal.h>.12829 The **timespec** structure shall be defined as described in <time.h>.12830 The <sys/select.h> header shall define the **fd_set** type as a structure that includes at least the
12831 following member:

12832 long fds_bits[] Bit mask for open file descriptions.

12833 Each of the following may be declared as a function, or defined as a macro, or both:

12834 void FD_CLR(int *fd*, fd_set **fdset*)12835 Clears the bit for the file descriptor *fd* in the file descriptor set *fdset*.12836 int FD_ISSET(int *fd*, fd_set **fdset*)12837 Returns a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set by
12838 *fdset*, and 0 otherwise.12839 void FD_SET(int *fd*, fd_set **fdset*)12840 Sets the bit for the file descriptor *fd* in the file descriptor set *fdset*.12841 void FD_ZERO(fd_set **fdset*)12842 Initializes the file descriptor set *fdset* to have zero bits for all file descriptors.12843 **FD_SETSIZE**12844 Maximum number of file descriptors in an **fd_set** structure.12845 If implemented as macros, these may evaluate their arguments more than once, so applications
12846 should ensure that the arguments they supply are never expressions with side effects.12847 The following shall be declared as functions and may also be defined as macros. Function
12848 prototypes shall be provided for use with an ISO C standard compiler.12849 int pselect(int, fd_set *, fd_set *, fd_set *, const struct timespec *,
12850 const sigset_t *);

12851 int select(int, fd_set *, fd_set *, fd_set *, struct timeval *);

12852 Inclusion of the <sys/select.h> header may make visible all symbols from the headers
12853 <signal.h>, <sys/time.h>, and <time.h>.

12854 **APPLICATION USAGE**

12855 None.

12856 **RATIONALE**

12857 None.

12858 **FUTURE DIRECTIONS**

12859 None.

12860 **SEE ALSO**

12861 <signal.h>, <sys/time.h>, <sys/types.h>, <time.h>, the System Interfaces volume of
12862 IEEE Std. 1003.1-200x, *pselect()*, *select()*

12863 **CHANGE HISTORY**

12864 First released in Issue 6. Derived from IEEE Std. 1003.1g-2000.

12865 **NAME**

12866 sys/sem.h — XSI semaphore facility

12867 **SYNOPSIS**

12868 XSI #include <sys/sem.h>

12869

12870 **DESCRIPTION**

12871 The <sys/sem.h> header shall define the following constants and structures.

12872 Semaphore operation flags:

12873 SEM_UNDO Set up adjust on exit entry.

12874 Command definitions for the *semctl()* function shall be provided as follows:

12875 GETNCNT Get *semncnt*.

12876 GETPID Get *sempid*.

12877 GETVAL Get *semval*.

12878 GETALL Get all cases of *semval*.

12879 GETZCNT Get *semzcnt*.

12880 SETVAL Set *semval*.

12881 SETALL Set all cases of *semval*.

12882 The **semid_ds** structure shall contain the following members:

12883 struct ipc_perm sem_perm Operation permission structure.

12884 unsigned short sem_nsems Number of semaphores in set.

12885 time_t sem_otime Last *semop()* time.

12886 time_t sem_ctime Last time changed by *semctl()*.

12887 The **pid_t**, **time_t**, **key_t**, and **size_t** types shall be defined as described in <sys/types.h>.

12888 A semaphore shall be represented by an anonymous structure containing the following
12889 members:

12890 unsigned short semval Semaphore value.

12891 pid_t sempid Process ID of last operation.

12892 unsigned short semncnt Number of processes waiting for *semval*
12893 to become greater than current value.

12894 unsigned short semzcnt Number of processes waiting for *semval*
12895 to become 0.

12896 The **sembuf** structure shall contain the following members:

12897 unsigned short sem_num Semaphore number.

12898 short sem_op Semaphore operation.

12899 short sem_flg Operation flags.

12900 The following shall be declared as functions and may also be defined as macros. Function
12901 prototypes shall be provided for use with an ISO C standard compiler.

12902 int semctl(int, int, int, ...);

12903 int semget(key_t, int, int);

12904 int semop(int, struct sembuf *, size_t);

12905 In addition, all of the symbols from **<sys/ipc.h>** shall be defined when this header is included.

12906 **APPLICATION USAGE**

12907 None.

12908 **RATIONALE**

12909 None.

12910 **FUTURE DIRECTIONS**

12911 None.

12912 **SEE ALSO**

12913 **<sys/types.h>**, *semctl()*, *semget()*, *semop()*

12914 **CHANGE HISTORY**

12915 First released in Issue 2. Derived from System V Release 2.0.

12916 **Issue 4**

12917 The function declarations in this header are expanded to full ISO C standard prototypes.

12918 Reference to the **<sys/types.h>** header is added for the definitions of **pid_t**, **time_t**, **key_t**, and
12919 **size_t**.

12920 A statement is added indicating that all symbols in **<sys/ipc.h>** are defined when this header is
12921 included.

12922 **NAME**

12923 sys/shm.h — XSI shared memory facility

12924 **SYNOPSIS**

12925 XSI `#include <sys/shm.h>`

12926

12927 **DESCRIPTION**

12928 The <sys/shm.h> header shall define the following symbolic constants:

12929 SHM_RDONLY Attach read-only (else read-write).

12930 SHM_RND Round attach address to SHMLBA.

12931 The <sys/shm.h> header shall define the following symbolic value:

12932 SHMLBA Segment low boundary address multiple.

12933 The following data types shall be defined through **typedef**:

12934 **shmatt_t** Unsigned integer used for the number of current attaches that must be able to
12935 store values at least as large as a type **unsigned short**.

12936 The **shmid_ds** structure shall contain the following members:

12937 struct ipc_perm shm_perm Operation permission structure.

12938 size_t shm_segsz Size of segment in bytes.

12939 pid_t shm_lpid Process ID of last shared memory operation.

12940 pid_t shm_cpid Process ID of creator.

12941 shmatt_t shm_nattch Number of current attaches.

12942 time_t shm_atime Time of last *shmat()*.

12943 time_t shm_dtime Time of last *shmdt()*.

12944 time_t shm_ctime Time of last change by *shmctl()*.

12945 The **pid_t**, **time_t**, **key_t**, and **size_t** types shall be defined as described in <sys/types.h>.

12946 The following shall be declared as functions and may also be defined as macros. Function
12947 prototypes shall be provided for use with an ISO C standard compiler.

12948 void *shmat(int, const void *, int);

12949 int shmctl(int, int, struct shmid_ds *);

12950 int shmdt(const void *);

12951 int shmget(key_t, size_t, int);

12952 In addition, all of the symbols from <sys/ipc.h> shall be defined when this header is included.

12953 **APPLICATION USAGE**

12954 None.

12955 **RATIONALE**

12956 None.

12957 **FUTURE DIRECTIONS**

12958 None.

12959 **SEE ALSO**

12960 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *shmat()*, *shmctl()*, *shmdt()*,

12961 *shmget()*

12962 **CHANGE HISTORY**

12963 First released in Issue 2. Derived from System V Release 2.0.

12964 **Issue 4**

12965 The function declarations in this header are expanded to full ISO C standard prototypes.

12966 Reference to the <sys/types.h> header is added for the definitions of **pid_t**, **time_t**, **key_t**, and
12967 **size_t**.

12968 A statement is added indicating that all symbols in <sys/ipc.h> are defined when this header is
12969 included.

12970 **Issue 5**

12971 The type of *shm_segsz* is changed from **int** to **size_t**.

12972 **NAME**

12973 sys/socket.h — main sockets header

12974 **SYNOPSIS**

12975 #include <sys/socket.h>

12976 **DESCRIPTION**

12977 The <sys/socket.h> header shall make available the type, **socklen_t**, which is an opaque integer
 12978 type of length of at least 32 bits; see APPLICATION USAGE.

12979 The <sys/socket.h> header shall define the unsigned integer type **sa_family_t**.

12980 The <sys/socket.h> header shall define the **sockaddr** structure that includes at least the
 12981 following members:

12982 sa_family_t sa_family Address family.
 12983 char sa_data[] Socket address (variable-length data).

12984 The **sockaddr** structure is used to define a socket address which is used in the *bind()*, *connect()*,
 12985 *getpeername()*, *getsockname()*, *recvfrom()*, and *sendto()* functions.

12986 The <sys/socket.h> header shall define the **sockaddr_storage** structure. This structure shall be:

- 12987 • Large enough to accommodate all supported protocol-specific address structures
- 12988 • Aligned at an appropriate boundary so that pointers to it can be cast as pointers to protocol-
 12989 specific address structures and used to access the fields of those structures without
 12990 alignment problems

12991 The **sockaddr_storage** structure shall contain at least the following members:

12992 sa_family_t ss_family

12993 When a **sockaddr_storage** structure is cast as a **sockaddr** structure, the *ss_family* field of the
 12994 **sockaddr_storage** structure maps onto the *sa_family* field of the **sockaddr** structure. When a
 12995 **sockaddr_storage** structure is cast as a protocol-specific address structure, the *ss_family* field
 12996 maps onto a field of that structure that is of type **sa_family_t** and that identifies the protocol's
 12997 address family.

12998 The <sys/socket.h> header shall define the **msghdr** structure that includes at least the following
 12999 members:

13000 void *msg_name Optional address.
 13001 socklen_t msg_namelen Size of address.
 13002 struct iovec *msg_iov Scatter/gather array.
 13003 int msg_iovlen Members in msg_iov.
 13004 void *msg_control Ancillary data; see below.
 13005 socklen_t msg_controllen Ancillary data buffer len.
 13006 int msg_flags Flags on received message.

13007 The **msghdr** structure is used to minimize the number of directly supplied parameters to the
 13008 *recvmsg()* and *sendmsg()* functions. This structure is used as a *value=result* parameter in the
 13009 *recvmsg()* function and *value* only for the *sendmsg()* function.

13010 The **iovec** structure shall be defined through **typedef** as described in <sys/uio.h>.

13011 The <sys/socket.h> header shall define the **cmsghdr** structure that includes at least the following
 13012 members:

13013 socklen_t cmsg_len Data byte count, including the cmsghdr.
 13014 int cmsg_level Originating protocol.

13015 int *cmsg_type* Protocol-specific type.

13016 The **cmsghdr** structure is used for storage of ancillary data object information.

13017 Ancillary data consists of a sequence of pairs, each consisting of a **cmsghdr** structure followed
13018 by a data array. The data array contains the ancillary data message, and the **cmsghdr** structure
13019 contains descriptive information that allows an application to correctly parse the data.

13020 The values for *cmsg_level* shall be legal values for the *level* argument to the *getsockopt()* and
13021 *setsockopt()* functions. The system documentation shall specify the *cmsg_type* definitions for the
13022 supported protocols.

13023 Ancillary data is also possible at the socket level. The **<sys/socket.h>** header defines the
13024 following macro for use as the *cmsg_type* value when *cmsg_level* is SOL_SOCKET:

13025 SCM_RIGHTS Indicates that the data array contains the access rights to be sent or
13026 received.

13027 The **<sys/socket.h>** header defines the following macros to gain access to the data arrays in the
13028 ancillary data associated with a message header:

13029 MSG_DATA(*cmsg*)
13030 If the argument is a pointer to a **cmsghdr** structure, this macro returns an unsigned
13031 character pointer to the data array associated with the **cmsghdr** structure.

13032 MSG_NXTHDR(*mhdr, cmsg*)
13033 If the first argument is a pointer to a **msghdr** structure and the second argument is a pointer
13034 to a **cmsghdr** structure in the ancillary data pointed to by the *msg_control* field of that
13035 **msghdr** structure, this macro returns a pointer to the next **cmsghdr** structure, or a null
13036 pointer if this structure is the last **cmsghdr** in the ancillary data.

13037 MSG_FIRSTHDR(*mhdr*)
13038 If the argument is a pointer to a **msghdr** structure, this macro returns a pointer to the first
13039 **cmsghdr** structure in the ancillary data associated with this **msghdr** structure, or a null
13040 pointer if there is no ancillary data associated with the **msghdr** structure.

13041 The **<sys/socket.h>** header shall define the **linger** structure that includes at least the following
13042 members:

13043 int *l_onoff* Indicates whether linger option is enabled.
13044 int *l_linger* Linger time, in seconds.

13045 The **<sys/socket.h>** header shall define the following macros, with distinct integral values:

13046 SOCK_DGRAM Datagram socket.
13047 SOCK_STREAM Byte-stream socket.
13048 SOCK_SEQPACKET Sequenced-packet socket.

13049 The **<sys/socket.h>** header shall define the following macro for use as the *level* argument of
13050 *setsockopt()* and *getsockopt()*.

13051 SOL_SOCKET Options to be accessed at socket level, not protocol level.

13052 The **<sys/socket.h>** header shall define the following macros, with distinct integral values, for
13053 use as the *option_name* argument in *getsockopt()* or *setsockopt()* calls:

13054 SO_ACCEPTCONN Socket is accepting connections.
13055 SO_BROADCAST Transmission of broadcast messages is supported.

13056	SO_DEBUG	Debugging information is being recorded.
13057	SO_DONTROUTE	Bypass normal routing.
13058	SO_ERROR	Socket error status.
13059	SO_KEEPALIVE	Connections are kept alive with periodic messages.
13060	SO_LINGER	Socket lingers on close.
13061	SO_OOBINLINE	Out-of-band data is transmitted in line.
13062	SO_RCVBUF	Receive buffer size.
13063	SO_RCVLOWAT	Receive “low water mark”.
13064	SO_RCVTIMEO	Receive timeout.
13065	SO_REUSEADDR	Reuse of local addresses is supported.
13066	SO_SNDBUF	Send buffer size.
13067	SO_SNDLOWAT	Send “low water mark”.
13068	SO_SNDTIMEO	Send timeout.
13069	SO_TYPE	Socket type.
13070	The <sys/socket.h> header shall define the following macro as the maximum <i>backlog</i> queue	
13071	length which may be specified by the <i>backlog</i> field of the <i>listen()</i> function:	
13072	SOMAXCONN	The maximum <i>backlog</i> queue length.
13073	The <sys/socket.h> header shall define the following macros, with distinct integral values, for	
13074	use as the valid values for the <i>msg_flags</i> field in the msghdr structure, or the <i>flags</i> parameter in	
13075	<i>recvfrom()</i> , <i>recvmsg()</i> , <i>sendmsg()</i> , or <i>sendto()</i> calls:	
13076	MSG_CTRUNC	Control data truncated.
13077	MSG_DONTROUTE	Send without using routing tables.
13078	MSG_EOR	Terminates a record (if supported by the protocol).
13079	MSG_OOB	Out-of-band data.
13080	MSG_PEEK	Leave received data in queue.
13081	MSG_TRUNC	Normal data truncated.
13082	MSG_WAITALL	Wait for complete message.
13083	The <sys/socket.h> header shall define the following macros, with distinct integral values:	
13084	AF_UNIX	UNIX domain sockets.
13085	AF_UNSPEC	Unspecified .
13086	AF_INET	Internet domain sockets for use with IPv4 addresses.
13087	AF_INET6	Internet domain sockets for use with IPv6 addresses.
13088	The <sys/socket.h> header shall define the following macros, with distinct integral values:	
13089	SHUT_RD	Disables further receive operations.
13090	SHUT_WR	Disables further send operations.

13091 SHUT_RDWR Disables further send and receive operations.

13092 The following are declared as functions, and may also be defined as macros. Function prototypes
13093 shall be provided for use with an ISO C standard compiler.

```

13094       int        accept(int, struct sockaddr *restrict, socklen_t *restrict);
13095       int        bind(int, const struct sockaddr *, socklen_t);
13096       int        connect(int, const struct sockaddr *, socklen_t);
13097       int        getpeername(int, struct sockaddr *restrict, socklen_t *);
13098       int        getsockname(int, struct sockaddr *restrict, socklen_t *);
13099       int        getsockopt(int, int, int, void *restrict, socklen_t *restrict);
13100       int        listen(int, int);
13101       ssize_t    recv(int, void *, size_t, int);
13102       ssize_t    recvfrom(int, void *restrict, size_t, int,
13103                    struct sockaddr *restrict, socklen_t *restrict);
13104       ssize_t    recvmsg(int, struct msghdr *, int);
13105       ssize_t    send(int, const void *, size_t, int);
13106       ssize_t    sendmsg(int, const struct msghdr *, int);
13107       ssize_t    sendto(int, const void *, size_t, int, const struct sockaddr *,
13108                    socklen_t);
13109       int        setsockopt(int, int, int, const void *, socklen_t);
13110       int        shutdown(int, int);
13111       int        socket(int, int, int);
13112       int        socketpair(int, int, int, int);

```

13113 Inclusion of **<sys/socket.h>** may also make visible all symbols from **<sys/uid.h>**.

13114 APPLICATION USAGE

13115 To forestall portability problems, it is recommended that applications not use values larger than
13116 $2^{32} - 1$ for the **socklen_t** type.

13117 The **sockaddr_storage** structure solves the problem of declaring storage for automatic variables
13118 which is both large enough and aligned enough for storing the socket address data structure of
13119 any family. For example, code with a file descriptor and without the context of the address
13120 family can pass a pointer to a variable of this type, where a pointer to a socket address structure
13121 is expected in calls such as *getpeername()*, and determine the address family by accessing the
13122 received content after the call.

13123 An example implementation design of such a data structure would be as follows:

```

13124       /*
13125        *   Desired design of maximum size and alignment.
13126        */
13127       #define _SS_MAXSIZE 128
13128       /* Implementation-defined maximum size. */
13129       #define _SS_ALIGNSIZE (sizeof(int64_t))
13130       /* Implementation-defined desired alignment. */
13131       /*
13132        *   Definitions used for sockaddr_storage structure paddings design.
13133        */
13134       #define _SS_PAD1SIZE (_SS_ALIGNSIZE - sizeof(sa_family_t))
13135       #define _SS_PAD2SIZE (_SS_MAXSIZE - (sizeof(sa_family_t)+
13136                                            _SS_PAD1SIZE + _SS_ALIGNSIZE))
13137       struct sockaddr_storage {
13138           sa_family_t   ss_family; /* Address family. */

```



```

13139  /*
13140  *  Following fields are implementation-defined. */
13141  */
13142  char _ss_pad1[_SS_PAD1SIZE];
13143  /* 6-byte pad; this is to make implementation-defined
13144  pad up to alignment field that follows explicit in
13145  the data structure. */
13146  int64_t _ss_align; /* Field to force desired structure
13147  storage alignment. */
13148  char _ss_pad2[_SS_PAD2SIZE];
13149  /* 112-byte pad to achieve desired size,
13150  _SS_MAXSIZE value minus size of ss_family
13151  __ss_pad1, __ss_align fields is 112. */
13152  };

```

13153 The above example illustrates a data structure which aligns on a 64-bit boundary. An
13154 implementation-defined field `_ss_align` along `_ss_pad1` is used to force a 64-bit alignment which
13155 covers proper alignment good enough for needs of `sockaddr_in6` (IPv6), `sockaddr_in` (IPv4)
13156 address data structures. The size of padding fields `_ss_pad1` depends on the chosen alignment
13157 boundary. The size of padding field `_ss_pad2` depends on the value of overall size chosen for the
13158 total size of the structure. This size and alignment are represented in the above example by
13159 implementation-defined (not required) constants `_SS_MAXSIZE` (chosen value 128) and
13160 `_SS_ALIGNMENT` (with chosen value 8). Constants `_SS_PAD1SIZE` (derived value 6) and
13161 `_SS_PAD2SIZE` (derived value 112) are also for illustration and not required. The
13162 implementation-defined definitions and structure field names above start with an underscore to
13163 denote implementation private name space. Portable code is not expected to access or reference
13164 those fields or constants.

13165 RATIONALE

13166 None.

13167 FUTURE DIRECTIONS

13168 None.

13169 SEE ALSO

13170 <sys/uio.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, `accept()`, `bind()`, `connect()`,
13171 `getpeername()`, `getsockname()`, `getsockopt()`, `listen()`, `recv()`, `recvfrom()`, `recvmsg()`, `send()`,
13172 `sendmsg()`, `sendto()`, `setsockopt()`, `shutdown()`, `socket()`, `socketpair()`

13173 CHANGE HISTORY

13174 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

13175 The **restrict** keyword is added to the prototypes for `accept()`, `getpeername()`, `getsockname()`,
13176 `getsockopt()`, and `recvfrom()`.

13177 NAME

13178 sys/stat.h — data returned by the stat() function

13179 SYNOPSIS

13180 #include <sys/stat.h>

13181 DESCRIPTION

13182 XSI The <sys/stat.h> header shall define the structure of the data returned by the functions *fstat()*,
13183 *lstat()*, and *stat()*.

13184 The **stat** structure shall contain at least the following members:

13185	dev_t	st_dev	ID of device containing file.
13186	ino_t	st_ino	File serial number.
13187	mode_t	st_mode	Mode of file (see below).
13188	nlink_t	st_nlink	Number of hard links to the file.
13189	uid_t	st_uid	User ID of file.
13190	gid_t	st_gid	Group ID of file.
13191 XSI	dev_t	st_rdev	Device ID (if file is character or block special).
13192	off_t	st_size	For regular files, the file size in bytes.
13193			For symbolic links, the length in bytes of the
13194			path name contained in the symbolic link.
13195			For other file types, the use of this field is
13196			unspecified
13197	time_t	st_atime	Time of last access.
13198	time_t	st_mtime	Time of last data modification.
13199	time_t	st_ctime	Time of last status change.
13200 XSI	blksize_t	st_blksize	A file system-specific preferred I/O block size for
13201			this object. In some file system types, this may
13202			vary from file to file.
13203	blkcnt_t	st_blocks	Number of blocks allocated for this object.
13204			

13205 File serial number and device ID taken together uniquely identify the file within the system. The
13206 **blkcnt_t**, **blksize_t**, **dev_t**, **ino_t**, **mode_t**, **nlink_t**, **uid_t**, **gid_t**, **off_t**, and **time_t** types shall be
13207 defined as described in <sys/types.h>. Times shall be given in seconds since the Epoch.

13208 Unless otherwise specified, the structure members *st_mode*, *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atime*,
13209 *st_ctime*, and *st_mtime* shall have meaningful values for all file types defined in
13210 IEEE Std. 1003.1-200x.

13211 For symbolic links, the *st_mode* member shall contain meaningful information, which can be
13212 used with the file type macros described below, that take a *mode* argument. The *st_size* member
13213 shall contain the length, in bytes, of the path name contained in the symbolic link. File mode bits
13214 and the contents of the remaining members of the **stat** structure are unspecified. The value
13215 returned in the *st_size* field shall be the length of the contents of the symbolic link, and shall not
13216 count a trailing null if one is present.

13217 The following symbolic names for the values of type *mode_t* shall also be defined.

13218 File type:

13219 XSI	S_IFMT	Type of file.
13220	S_IFBLK	Block special.
13221	S_IFCHR	Character special.

13222	S_IFIFO	FIFO special.
13223	S_IFREG	Regular.
13224	S_IFDIR	Directory.
13225	S_IFLNK	Symbolic link.
13226	S_IFSOCK	Socket.
13227	File mode bits:	
13228	S_IRWXU	Read, write, execute/search by owner.
13229	S_IRUSR	Read permission, owner.
13230	S_IWUSR	Write permission, owner.
13231	S_IXUSR	Execute/search permission, owner.
13232	S_IRWXG	Read, write, execute/search by group.
13233	S_IRGRP	Read permission, group.
13234	S_IWGRP	Write permission, group.
13235	S_IXGRP	Execute/search permission, group.
13236	S_IRWXO	Read, write, execute/search by others.
13237	S_IROTH	Read permission, others.
13238	S_IWOTH	Write permission, others.
13239	S_IXOTH	Execute/search permission, others.
13240	S_ISUID	Set-user-ID on execution.
13241	S_ISGID	Set-group-ID on execution.
13242 XSI	S_ISVTX	On directories, restricted deletion flag.
13243	The bits defined by S_IRUSR , S_IWUSR , S_IXUSR , S_IRGRP , S_IWGRP , S_IXGRP , S_IROTH ,	
13244 XSI	S_IWOTH , S_IXOTH , S_ISUID , S_ISGID , and S_ISVTX shall be unique.	
13245	S_IRWXU is the bitwise-inclusive OR of S_IRUSR , S_IWUSR , and S_IXUSR .	
13246	S_IRWXG is the bitwise-inclusive OR of S_IRGRP , S_IWGRP , and S_IXGRP .	
13247	S_IRWXO is the bitwise-inclusive OR of S_IROTH , S_IWOTH , and S_IXOTH .	
13248	Implementations may OR other implementation-defined bits into S_IRWXU , S_IRWXG , and	
13249	S_IRWXO , but they shall not overlap any of the other bits defined in this volume of	
13250	IEEE Std. 1003.1-200x. The <i>file permission bits</i> are defined to be those corresponding to the	
13251	bitwise-inclusive OR of S_IRWXU , S_IRWXG , and S_IRWXO .	
13252	The following macros shall be provided to test whether a file is of the specified type. The value	
13253	<i>m</i> supplied to the macros is the value of <i>st_mode</i> from a stat structure. The macro shall evaluate	
13254	to a non-zero value if the test is true; 0 if the test is false.	
13255	S_ISBLK(<i>m</i>)	Test for a block special file.
13256	S_ISCHR(<i>m</i>)	Test for a character special file.
13257	S_ISDIR(<i>m</i>)	Test for a directory.

13258 **S_ISFIFO(*m*)** Test for a pipe or FIFO special file.
 13259 **S_ISREG(*m*)** Test for a regular file.
 13260 **S_ISLNK(*m*)** Test for a symbolic link.
 13261 **S_ISSOCK(*m*)** Test for a socket.

13262 The implementation may implement message queues, semaphores, or shared memory objects as
 13263 distinct file types. The following macros shall be provided to test whether a file is of the
 13264 specified type. The value of the *buf* argument supplied to the macros is a pointer to a **stat**
 13265 structure. The macro shall evaluate to a non-zero value if the specified object is implemented as
 13266 a distinct file type and the specified file type is contained in the **stat** structure referenced by *buf*.
 13267 Otherwise, the macro shall evaluate to zero.

13268 **S_TYPEISMQ(*buf*)** Test for a message queue.

13269 **S_TYPEISSEM(*buf*)** Test for a semaphore.

13270 **S_TYPEISSHM(*buf*)** Test for a shared memory object.

13271 TYP The implementation may implement typed memory objects as distinct file types, and the
 13272 following macro shall test whether a file is of the specified type. The value of the *buf* argument
 13273 supplied to the macros is a pointer to a **stat** structure. The macro shall evaluate to a non-zero
 13274 value if the specified object is implemented as a distinct file type and the specified file type is
 13275 contained in the **stat** structure referenced by *buf*. Otherwise, the macro shall evaluate to zero.

13276 **S_TYPEISTMO(*buf*)** Test macro for a typed memory object.

13277

13278 The following shall be declared as functions and may also be defined as macros. Function
 13279 prototypes shall be provided for use with an ISO C standard compiler.

```
13280        int     chmod(const char *, mode_t);
13281        int     fchmod(int, mode_t);
13282        int     fstat(int, struct stat *);
13283        int     isfdtype(int, int);
13284        int     lstat(const char *restrict, struct stat *restrict);
13285        int     mkdir(const char *, mode_t);
13286        int     mkfifo(const char *, mode_t);
13287 XSI        int     mknod(const char *, mode_t, dev_t);
13288        int     stat(const char *restrict, struct stat *restrict);
13289        mode_t  umask(mode_t);
```

13290 APPLICATION USAGE

13291 Use of the macros is recommended for determining the type of a file.

13292 RATIONALE

13293 A conforming C-language application must include **<sys/stat.h>** for functions that have
 13294 arguments or return values of type **mode_t**, so that symbolic values for that type can be used.
 13295 An alternative would be to require that these constants are also defined by including
 13296 **<sys/types.h>**.

13297 The **S_ISUID** and **S_ISGID** bits may be cleared on any write, not just on *open()*, as some historical
 13298 implementations do it.

13299 System calls that update the time entry fields in the **stat** structure must be documented by the
 13300 implementors. POSIX-conforming systems should not update the time entry fields for functions
 13301 listed in the System Interfaces volume of IEEE Std. 1003.1-200x unless the standard requires that
 13302 they do, except in the case of documented extensions to the standard.

- 13303 Note that *st_dev* must be unique within a Local Area Network (LAN) in a “system” made up of
13304 multiple computers’ file systems connected by a LAN.
- 13305 Networked implementations of a POSIX-conforming system must guarantee that all files visible
13306 within the file tree (including parts of the tree that may be remotely mounted from other
13307 machines on the network) on each individual processor are uniquely identified by the
13308 combination of the *st_ino* and *st_dev* fields.
- 13309 **FUTURE DIRECTIONS**
- 13310 None.
- 13311 **SEE ALSO**
- 13312 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *chmod()*, *fchmod()*, *fstat()*,
13313 *lstat()*, *mkdir()*, *mkfifo()*, *mknod()*, *stat()*, *umask()*
- 13314 **CHANGE HISTORY**
- 13315 First released in Issue 1. Derived from Issue 1 of the SVID.
- 13316 **Issue 4**
- 13317 Reference to the <sys/types.h> header is added for the definitions of **dev_t**, **ino_t**, **mode_t**,
13318 **nlink_t**, **uid_t**, **gid_t**, **off_t**, and **time_t**. This has been marked as an extension.
- 13319 References to the S_IREAD, S_IWRITE, S_IEXEC file, and S_ISVTX modes are removed.
- 13320 The descriptions of the members of the **stat** structure in the DESCRIPTION are corrected.
- 13321 The following changes are incorporated for alignment with the ISO POSIX-1 standard:
- 13322 • The function declarations in this header are expanded to full ISO C standard prototypes.
 - 13323 • The DESCRIPTION is expanded to include:
 - 13324 — How files are uniquely identified within the system
 - 13325 — Times are given in units of seconds since the Epoch
 - 13326 — Rules governing the definition and use of the file mode bits
 - 13327 — Usage of the file type test macros
- 13328 **Issue 4, Version 2**
- 13329 The following changes are incorporated for X/OPEN UNIX conformance:
- 13330 • The *st_blksize* and *st_blocks* members are added to the **stat** structure.
 - 13331 • The S_IFLINK value of S_IFMT is defined.
 - 13332 • The S_ISVTX file mode bit and the S_ISLNK file type test macro is defined.
 - 13333 • The *fchmod()*, *lstat()*, and *mknod()* functions are added to the list of functions declared in this
13334 header.
- 13335 **Issue 5**
- 13336 The DESCRIPTION is updated for alignment with POSIX Realtime Extension.
- 13337 The type of *st_blksize* is changed from **long** to **blksize_t**; the type of *st_blocks* is changed from
13338 **long** to **blkcnt_t**.
- 13339 **Issue 6**
- 13340 The S_TYPEISMQ(), S_TYPEISSEM(), and S_TYPEISSHM() macros are unconditionally
13341 mandated.
- 13342 The Open Group corrigenda item U035/4 has been applied. In the DESCRIPTION, the types
13343 **blksize_t** and **blkcnt_t** have been described.

13344 The following new requirements on POSIX implementations derive from alignment with the
13345 Single UNIX Specification:

- 13346 • The **dev_t**, **ino_t**, **mode_t**, **nlink_t**, **uid_t**, **gid_t**, **off_t**, and **time_t** types are mandated.

13347 The *isfdtype()* function, S_IFSOCK, and S_ISSOCK are added for sockets.

13348 The description of **stat** structure members is changed to reflect contents when file type is a
13349 symbolic link.

13350 The test macro S_TYPEISTMO is added for alignment with IEEE Std. 1003.1j-2000. |

13351 The **restrict** keyword is added to the prototypes for *lstat()* and *stat()*. |

13352 **NAME**

13353 sys/statvfs.h — VFS File System information structure

13354 **SYNOPSIS**

13355 XSI #include <sys/statvfs.h>

13356

13357 **DESCRIPTION**

13358 The <sys/statvfs.h> header shall define the **statvfs** structure that includes at least the following
 13359 members:

13360	unsigned long	f_bsize	File system block size.
13361	unsigned long	f_frsize	Fundamental file system block size.
13362	fsblkcnt_t	f_blocks	Total number of blocks on file system in units of <i>f_frsize</i> .
13363	fsblkcnt_t	f_bfree	Total number of free blocks.
13364	fsblkcnt_t	f_bavail	Number of free blocks available to non-privileged process.
13365			
13366	fsfilcnt_t	f_files	Total number of file serial numbers.
13367	fsfilcnt_t	f_ffree	Total number of free file serial numbers.
13368	fsfilcnt_t	f_favail	Number of file serial numbers available to non-privileged process.
13369			
13370	unsigned long	f_fsid	File system ID.
13371	unsigned long	f_flag	Bit mask of <i>f_flag</i> values.
13372	unsigned long	f_namemax	Maximum file name length.

13373 The **fsblkcnt_t** and **fsfilcnt_t** types shall be defined as described in <sys/types.h>.

13374 The following flags for the *f_flag* member shall be defined:

13375	ST_RDONLY	Read-only file system.
13376	ST_NOSUID	Does not support setuid/setgid semantics.

13377 The <sys/statvfs.h> header shall declare the following functions which may also be defined as
 13378 macros. Function prototypes shall be provided for use with an ISO C standard compiler.

```
13379 int statvfs(const char *restrict, struct statvfs *restrict);
13380 int fstatvfs(int, struct statvfs *);
```

13381 **APPLICATION USAGE**

13382 None.

13383 **RATIONALE**

13384 None.

13385 **FUTURE DIRECTIONS**

13386 None.

13387 **SEE ALSO**

13388 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *fstatvfs()*, *statvfs()*

13389 **CHANGE HISTORY**

13390 First released in Issue 4, Version 2.

13391 **Issue 5**

13392 The type of *f_blocks*, *f_bfree*, and *f_bavail* is changed from **unsigned long** to **fsblkcnt_t**; the type
 13393 of *f_files*, *f_ffree*, and *f_favail* is changed from **unsigned long** to **fsfilcnt_t**.

13394 **Issue 6**

13395 The Open Group corrigenda item U035/5 has been applied. In the DESCRIPTION, the types
13396 **fsblkcnt_t** and **fsfilcnt_t** have been described.

13397 The **restrict** keyword is added to the prototype for *statvfs()*.

13398 NAME

13399 sys/time.h — time types

13400 SYNOPSIS

13401 XSI #include <sys/time.h>

13402

13403 DESCRIPTION

13404 The <sys/time.h> header shall define the **timeval** structure that includes at least the following
13405 members:

13406 time_t tv_sec Seconds.

13407 suseconds_t tv_usec Microseconds.

13408 The <sys/time.h> header shall define the **itimerval** structure that includes at least the following
13409 members:

13410 struct timeval it_interval Timer interval.

13411 struct timeval it_value Current value.

13412 The **time_t** and **suseconds_t** types shall be defined as described in <sys/types.h>.13413 The <sys/time.h> header shall define the **fd_set** type as a structure that includes at least the
13414 following member:

13415 long fds_bits[] Bit mask for open file descriptions.

13416 The <sys/time.h> header shall define the following values for the *which* argument of *getitimer()*
13417 and *setitimer()*:

13418 ITIMER_REAL Decrements in real time.

13419 ITIMER_VIRTUAL Decrements in process virtual time.

13420 ITIMER_PROF Decrements both in process virtual time and when the system is running
13421 on behalf of the process.

13422 Each of the following may be declared as a function, or defined as a macro, or both:

13423 void FD_CLR(int *fd*, fd_set **fdset*)13424 Clears the bit for the file descriptor *fd* in the file descriptor set *fdset*.13425 int FD_ISSET(int *fd*, fd_set **fdset*)13426 Returns a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set by
13427 *fdset*, and 0 otherwise.13428 void FD_SET(int *fd*, fd_set **fdset*)13429 Sets the bit for the file descriptor *fd* in the file descriptor set *fdset*.13430 void FD_ZERO(fd_set **fdset*)13431 Initializes the file descriptor set *fdset* to have zero bits for all file descriptors.

13432 FD_SETSIZE

13433 Maximum number of file descriptors in an **fd_set** structure.13434 If implemented as macros, these may evaluate their arguments more than once, so that
13435 arguments must never be expressions with side effects.13436 The following shall be declared as functions and may also be defined as macros. Function
13437 prototypes shall be provided for use with an ISO C standard compiler.

13438 int getitimer(int, struct itimerval *);

13439 int gettimeofday(struct timeval *, void *);

13440 int select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
13441 struct timeval *restrict);
13442 int setitimer(int, const struct itimerval *restrict,
13443 struct itimerval *restrict);
13444 int utimes(const char *, const struct timeval [2]); (**LEGACY**)

13445 Inclusion of the <sys/time.h> header may make visible all symbols from the <sys/select.h>
13446 header.

13447 **APPLICATION USAGE**

13448 None.

13449 **RATIONALE**

13450 None.

13451 **FUTURE DIRECTIONS**

13452 None.

13453 **SEE ALSO**

13454 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *getitimer()*, *gettimeofday()*,
13455 *select()*, *setitimer()*

13456 **CHANGE HISTORY**

13457 First released in Issue 4, Version 2.

13458 **Issue 5**

13459 The type of *tv_usec* is changed from **long** to **suseconds_t**.

13460 **Issue 6**

13461 The **restrict** keyword is added to the prototypes for *select()* and *setitimer()*.

13462 The note is added that inclusion of this header may also make symbols visible from
13463 <sys/socket.h>.

13464 **NAME**

13465 sys/timeb.h — additional definitions for date and time

13466 **SYNOPSIS**

13467 XSI #include <sys/timeb.h>

13468

13469 **DESCRIPTION**

13470 The <sys/timeb.h> header shall define the **timeb** structure that includes at least the following
 13471 members:

13472	time_t	time	The seconds portion of the current time.
13473	unsigned short	millitm	The milliseconds portion of the current time.
13474	short	timezone	The local timezone in minutes west of Greenwich.
13475	short	dstflag	TRUE if Daylight Savings Time is in effect.

13476 The **time_t** type shall be defined as described in <sys/types.h>.

13477 The <sys/timeb.h> header shall declare the following as a function which may also be defined as
 13478 a macro. Function prototypes shall be provided for use with an ISO C standard compiler.

13479 int ftime(struct timeb *); (**LEGACY**)

13480 **APPLICATION USAGE**

13481 None.

13482 **RATIONALE**

13483 None.

13484 **FUTURE DIRECTIONS**

13485 None.

13486 **SEE ALSO**

13487 <sys/types.h>, <time.h>

13488 **CHANGE HISTORY**

13489 First released in Issue 4, Version 2.

13490 **Issue 6**

13491 The *ftime()* function is marked LEGACY.

13492 **NAME**13493 `sys/times.h` — file access and modification times structure13494 **SYNOPSIS**13495 `#include <sys/times.h>`13496 **DESCRIPTION**13497 The **<sys/times.h>** header shall define the structure **tms**, which is returned by *times()* and
13498 includes at least the following members:13499 `clock_t tms_utime` User CPU time.13500 `clock_t tms_stime` System CPU time.13501 `clock_t tms_cutime` User CPU time of terminated child processes.13502 `clock_t tms_cstime` System CPU time of terminated child processes.13503 The **clock_t** type shall be defined as described in **<sys/types.h>**.13504 The following shall be declared as a function and may also be defined as a macro. Function
13505 prototypes shall be provided for use with an ISO C standard compiler.13506 `clock_t times(struct tms *);`13507 **APPLICATION USAGE**

13508 None.

13509 **RATIONALE**

13510 None.

13511 **FUTURE DIRECTIONS**

13512 None.

13513 **SEE ALSO**13514 **<sys/types.h>**, the System Interfaces volume of IEEE Std. 1003.1-200x, *times()*13515 **CHANGE HISTORY**

13516 First released in Issue 1. Derived from Issue 1 of the SVID.

13517 **Issue 4**13518 Reference to the **<sys/types.h>** header is added for the definitions of **clock_t**.13519 This issue states that the *times()* function can also be defined as a macro.

13520 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 13521
- The function declarations in this header are expanded to full ISO C standard prototypes.

13522 **NAME**

13523 sys/types.h — data types

13524 **SYNOPSIS**

13525 #include <sys/types.h>

13526 **DESCRIPTION**

13527 The <sys/types.h> header shall include definitions for at least the following types:

13528	blkcnt_t	Used for file block counts.
13529	blksize_t	Used for block sizes.
13530 XSI 13531	clock_t	Used for system times in clock ticks or CLOCKS_PER_SEC; see <time.h>.
13532 TMR	clockid_t	Used for clock ID type in the clock and timer functions.
13533	dev_t	Used for device IDs.
13534 XSI	fsblkcnt_t	Used for file system block counts.
13535 XSI	fsfilcnt_t	Used for file system file counts.
13536	gid_t	Used for group IDs.
13537 XSI 13538	id_t	Used as a general identifier; can be used to contain at least a pid_t , uid_t , or gid_t .
13539	ino_t	Used for file serial numbers.
13540 XSI	key_t	Used for XSI interprocess communication.
13541	mode_t	Used for some file attributes.
13542	nlink_t	Used for link counts.
13543	off_t	Used for file sizes.
13544	pid_t	Used for process IDs and process group IDs.
13545 THR	pthread_attr_t	Used to identify a thread attribute object.
13546 BAR	pthread_barrier_t	Used to identify a barrier.
13547 BAR	pthread_barrierattr_t	Used to define a barrier attributes object.
13548 THR	pthread_cond_t	Used for condition variables.
13549 THR	pthread_condattr_t	Used to identify a condition attribute object.
13550 THR	pthread_key_t	Used for thread-specific data keys.
13551 THR	pthread_mutex_t	Used for mutexes.
13552 THR	pthread_mutexattr_t	Used to identify a mutex attribute object.
13553 THR	pthread_once_t	Used for dynamic package initialization.
13554 THR	pthread_rwlock_t	Used for read-write locks.
13555 THR	pthread_rwlockattr_t	Used for read-write lock attributes.
13556 SPI	pthread_spinlock_t	Used to identify a spin lock.
13557 THR	pthread_t	Used to identify a thread.

13558	size_t	Used for sizes of objects.
13559	ssize_t	Used for a count of bytes or an error indication.
13560 XSI	suseconds_t	Used for time in microseconds
13561	time_t	Used for time in seconds.
13562 TMR	timer_t	Used for timer ID returned by <i>timer_create()</i> .
13563	uid_t	Used for user IDs.
13564 XSI	useconds_t	Used for time in microseconds.

13565 All of the types shall be defined as arithmetic types of an appropriate length, with the following
 13566 exceptions:

13567 XSI	key_t
13568 THR	pthread_attr_t
13569 BAR	pthread_barrier_t
13570	pthread_barrierattr_t
13571 THR	pthread_cond_t
13572	pthread_condattr_t
13573	pthread_key_t
13574	pthread_mutex_t
13575	pthread_mutexattr_t
13576	pthread_once_t
13577	pthread_rwlock_t
13578	pthread_rwlockattr_t
13579 SPI	pthread_spinlock_t
13580 TRC	trace_attr_t
13581	trace_event_id_t
13582 TRC TEF	trace_event_set_t
13583 TRC	trace_id_t
13584	

13585 Additionally:

- 13586 • **blkcnt_t** and **off_t** shall be signed integer types.
- 13587 XSI • **fsblkcnt_t**, **fsfilcnt_t**, and **ino_t** shall be defined as unsigned integer types.
- 13588 • **size_t** shall be an unsigned integer type.
- 13589 • **blksize_t**, **pid_t**, and **ssize_t** shall be signed integer types.

13590 XSI The type **ssize_t** shall be capable of storing values at least in the range $[-1, \{SSIZE_MAX\}]$. The
 13591 type **useconds_t** shall be an unsigned integer type capable of storing values at least in the range
 13592 $[0, 1\ 000\ 000]$. The type **suseconds_t** shall be a signed integer type capable of storing values at
 13593 least in the range $[-1, 1\ 000\ 000]$.

13594 There are no defined comparison or assignment operators for the following types:

13595 THR	pthread_attr_t
13596 BAR	pthread_barrier_t
13597	pthread_barrierattr_t
13598 THR	pthread_cond_t
13599	pthread_condattr_t
13600	pthread_mutex_t
13601	pthread_mutexattr_t

13602 **pthread_rwlock_t**
 13603 **pthread_rwlockattr_t**
 13604 SPI **pthread_spinlock_t**
 13605 TRC **trace_attr_t**
 13606

13607 **APPLICATION USAGE**
 13608 None.

13609 **RATIONALE**
 13610 None.

13611 **FUTURE DIRECTIONS**
 13612 None.

13613 **SEE ALSO**
 13614 <time.h>

13615 **CHANGE HISTORY**
 13616 First released in Issue 1. Derived from Issue 1 of the SVID.

13617 **Issue 4**
 13618 The **clock_t** type is marked as an extension.
 13619 In the last paragraph of the DESCRIPTION, only the reference to type **key_t** is now marked as
 13620 an extension.
 13621 The following changes are incorporated for alignment with the ISO POSIX-1 standard:
 13622 • The data type **ssize_t** is added.
 13623 • The DESCRIPTION is expanded to indicate the required arithmetic types.

13624 **Issue 4, Version 2**
 13625 The **id_t** and **useconds_t** types are defined for X/OPEN UNIX conformance. The capability of
 13626 the **useconds_t** type is described.

13627 **Issue 5**
 13628 The **clockid_t** and **timer_t** types are defined for alignment with the POSIX Realtime Extension.
 13629 The types **blkcnt_t**, **blksize_t**, **fsblkcnt_t**, **fsfilcnt_t**, and **suseconds_t** are added.
 13630 Large File System extensions are added.
 13631 Updated for alignment with the POSIX Threads Extension.

13632 **Issue 6**
 13633 The **pthread_barrier_t**, **pthread_barrierattr_t**, and **pthread_spinlock_t** types are added for
 13634 alignment with IEEE Std. 1003.1j-2000.
 13635 The margin code is changed from XSI to THR for the **pthread_rwlock_t** and
 13636 **pthread_rwlockattr_t** types as Read-Write Locks have been absorbed into the POSIX Threads
 13637 option. The threads types are now marked THR.

13638 **NAME**

13639 sys/uio.h — definitions for vector I/O operations

13640 **SYNOPSIS**13641 XSI `#include <sys/uio.h>`

13642

13643 **DESCRIPTION**13644 The **<sys/uio.h>** header shall define the **iovec** structure that includes at least the following
13645 members:13646 `void *iov_base` Base address of a memory region for input or output.13647 `size_t iov_len` The size of the memory pointed to by `iov_base`.13648 The **<sys/uio.h>** header uses the **iovec** structure for scatter/gather I/O.13649 The **ssize_t** and **size_t** types shall be defined as described in **<sys/types.h>**.13650 The following shall be declared as functions and may also be defined as macros. Function
13651 prototypes shall be provided for use with an ISO C standard compiler.13652 `ssize_t readv(int, const struct iovec *, int);`13653 `ssize_t writev(int, const struct iovec *, int);`13654 **APPLICATION USAGE**13655 The implementation can put a limit on the number of scatter/gather elements which can be
13656 processed in one call. The symbol {IOV_MAX} defined in **<limits.h>** should always be used to
13657 learn about the limits instead of assuming a fixed value.13658 **RATIONALE**13659 Traditionally, the maximum number of scatter/gather elements the system can process in one
13660 call were described by the symbolic value {UIO_MAXIOV}. In IEEE Std. 1003.1-200x this value
13661 was replaced by the constant {IOV_MAX} which can be found in **<limits.h>**.13662 **FUTURE DIRECTIONS**

13663 None.

13664 **SEE ALSO**13665 **<limits.h>**, **<sys/types.h>**, the System Interfaces volume of IEEE Std. 1003.1-200x, `read()`, `write()`13666 **CHANGE HISTORY**

13667 First released in Issue 4, Version 2.

13668 **Issue 6**

13669 Text referring to scatter/gather I/O is added to the DESCRIPTION.

13670 **NAME**

13671 sys/un.h — definitions for UNIX domain sockets

13672 **SYNOPSIS**

13673 #include <sys/un.h>

13674 **DESCRIPTION**

13675 The <sys/un.h> header shall define the **sockaddr_un** structure that includes at least the
13676 following members:

13677 sa_family_t sun_family Address family.
13678 char sun_path[] Socket path name.

13679 The **sockaddr_un** structure is used to store addresses for UNIX domain sockets. Values of this
13680 type shall be cast by applications to **struct sockaddr** for use with socket functions.

13681 The **sa_family_t** type shall be defined as described in <sys/socket.h>.

13682 **APPLICATION USAGE**

13683 The size of *sun_path* has intentionally been left undefined. This is because different
13684 implementations use different sizes. For example, BSD4.3 uses a size of 108, and BSD4.4 uses a
13685 size of 104. Since most implementations originate from BSD versions, the size is typically in the
13686 range 92 to 108.

13687 Applications should not assume a particular length for *sun_path* or assume that it can hold
13688 `_POSIX_PATH_MAX` characters (255).

13689 **RATIONALE**

13690 None.

13691 **FUTURE DIRECTIONS**

13692 None.

13693 **SEE ALSO**

13694 <sys/socket.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *bind()*, *socket()*,
13695 *socketpair()*

13696 **CHANGE HISTORY**

13697 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

13698 **NAME**13699 `sys/utsname.h` — system name structure13700 **SYNOPSIS**13701 `#include <sys/utsname.h>`13702 **DESCRIPTION**13703 The **<sys/utsname.h>** header shall define the structure **utsname** which shall include at least the
13704 following members:

13705 `char sysname[]` Name of this implementation of the operating system.
13706 `char nodename[]` Name of this node within an implementation-defined
13707 communications network.
13708 `char release[]` Current release level of this implementation.
13709 `char version[]` Current version level of this release.
13710 `char machine[]` Name of the hardware type on which the system is running.

13711 The character arrays are of unspecified size, but the data stored in them shall be terminated by a
13712 null byte.

13713 The following shall be declared as a function and may also be defined as a macro:

13714 `int uname(struct utsname *);`13715 **APPLICATION USAGE**

13716 None.

13717 **RATIONALE**

13718 None.

13719 **FUTURE DIRECTIONS**

13720 None.

13721 **SEE ALSO**13722 The System Interfaces volume of IEEE Std. 1003.1-200x, `uname()`13723 **CHANGE HISTORY**

13724 First released in Issue 1. Derived from Issue 1 of the SVID.

13725 **Issue 4**

13726 The word “character” is replaced with the word “byte” in the DESCRIPTION.

13727 The function in this header can now also be defined as a macro.

13728 The following change is incorporated for alignment with the ISO C standard:

- 13729
- The function declarations in this header are expanded to full ISO C standard prototypes.

13730 **NAME**

13731 sys/wait.h — declarations for waiting

13732 **SYNOPSIS**

13733 #include <sys/wait.h>

13734 **DESCRIPTION**

13735 The <sys/wait.h> header shall define the following symbolic constants for use with *waitpid()*:

13736 WNOHANG Do not hang if no status is available; return immediately.

13737 WUNTRACED Report status of stopped child process.

13738 The <sys/wait.h> header shall define the following macros for analysis of process status values:

13739 WEXITSTATUS Return exit status.

13740 XSI WIFCONTINUED True if child has been continued

13741 WIFEXITED True if child exited normally.

13742 WIFSIGNALED True if child exited due to uncaught signal.

13743 WIFSTOPPED True if child is currently stopped.

13744 WSTOPSIG Return signal number that caused process to stop.

13745 WTERMSIG Return signal number that caused process to terminate.

13746 XSI The following symbolic constants shall be defined as possible values for the *options* argument to
13747 *waitid()*:

13748 WEXITED Wait for processes that have exited.

13749 WSTOPPED Status is returned for any child that has stopped upon receipt of a signal.

13750 WCONTINUED Status is returned for any child that was stopped and has been continued.

13751 WNOHANG Return immediately if there are no children to wait for.

13752 WNOWAIT Keep the process whose status is returned in *infp* in a waitable state.

13753 The type **idtype_t** shall be defined as an enumeration type whose possible values shall include
13754 at least the following:

13755 P_ALL

13756 P_PID

13757 P_PGID

13758

13759 The **id_t** and **pid_t** types shall be defined as described in <sys/types.h>.

13760 XSI The **siginfo_t** type shall be defined as described in <signal.h>.

13761 The **rusage** structure shall be defined as described in <sys/resource.h>.

13762 Inclusion of the <sys/wait.h> header may also make visible all symbols from <signal.h> and
13763 <sys/resource.h>.

13764 The following shall be declared as functions and may also be defined as macros. Function
13765 prototypes shall be provided for use with an ISO C standard compiler.

13766 pid_t wait(int *);

13767 XSI int waitid(idtype_t, id_t, siginfo_t *, int);

13768 pid_t waitpid(pid_t, int *, int);

13769 **APPLICATION USAGE**

13770 None.

13771 **RATIONALE**

13772 None.

13773 **FUTURE DIRECTIONS**

13774 None.

13775 **SEE ALSO**

13776 <signal.h>, <sys/resource.h>, <sys/types.h>, <sys/wait.h>, the System Interfaces volume of
13777 IEEE Std. 1003.1-200x, *wait()*, *waitid()*

13778 **CHANGE HISTORY**

13779 First released in Issue 3.

13780 Entry included for alignment with the POSIX.1-1988 standard.

13781 **Issue 4**

13782 Reference to the <sys/types.h> header is added for the definition of **pid_t** and marked as an
13783 extension.

13784 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 13785 • The function declarations in this header are expanded to full ISO C standard prototypes.

13786 **Issue 4, Version 2**

13787 The following changes are incorporated for X/OPEN UNIX conformance:

- 13788 • The WIFCONTINUED macro, the list of symbolic constants for the *options* argument to
13789 *waitid()*, and the description of the **idtype_t** enumeration type are added.
- 13790 • A statement is added indicated that inclusion of this header may also make visible constants
13791 from <signal.h> and <sys/resource.h>.
- 13792 • The *wait3()* and *waitid()* functions are added to the list of functions declared in this header.

13793 **Issue 6**

13794 The *wait3()* function is removed.

13795 **NAME**

13796 syslog — definitions for system error logging

13797 **SYNOPSIS**

13798 xSI #include <syslog.h>

13799

13800 **DESCRIPTION**13801 The <syslog.h> header shall define the following symbolic constants, zero or more of which may
13802 be OR'ed together to form the *logopt* option of *openlog()*:

13803 LOG_PID Log the process ID with each message.

13804 LOG_CONS Log to the system console on error.

13805 LOG_NDELAY Connect to syslog daemon immediately.

13806 LOG_ODELAY Delay open until *syslog()* is called.

13807 LOG_NOWAIT Do not wait for child processes.

13808 The following symbolic constants shall be defined as possible values of the *facility* argument to
13809 *openlog()*:

13810 LOG_KERN Reserved for message generated by the system.

13811 LOG_USER Message generated by a process.

13812 LOG_MAIL Reserved for message generated by mail system.

13813 LOG_NEWS Reserved for message generated by news system.

13814 LOG_UUCP Reserved for message generated by UUCP system.

13815 LOG_DAEMON Reserved for message generated by system daemon.

13816 LOG_AUTH Reserved for message generated by authorization daemon.

13817 LOG_CRON Reserved for message generated by the clock daemon.

13818 LOG_LPR Reserved for message generated by printer system.

13819 LOG_LOCAL0 Reserved for local use.

13820 LOG_LOCAL1 Reserved for local use.

13821 LOG_LOCAL2 Reserved for local use.

13822 LOG_LOCAL3 Reserved for local use.

13823 LOG_LOCAL4 Reserved for local use.

13824 LOG_LOCAL5 Reserved for local use.

13825 LOG_LOCAL6 Reserved for local use.

13826 LOG_LOCAL7 Reserved for local use.

13827 The following shall be declared as macros for constructing the *maskpri* argument to *setlogmask()*.13828 The following macros expand to an expression of type **int** when the argument *pri* is an
13829 expression of type **int**:13830 LOG_MASK(*pri*) A mask for priority *pri*.13831 The following constants shall be defined as possible values for the *priority* argument of *syslog()*:

13832 LOG_EMERG A panic condition was reported to all processes.
13833 LOG_ALERT A condition that should be corrected immediately.
13834 LOG_CRIT A critical condition.
13835 LOG_ERR An error message.
13836 LOG_WARNING A warning message.
13837 LOG_NOTICE A condition requiring special handling.
13838 LOG_INFO A general information message.
13839 LOG_DEBUG A message useful for debugging programs.
13840 The following shall be declared as functions and may also be defined as macros. Function
13841 prototypes shall be provided for use with an ISO C standard compiler.
13842 void closelog(void);
13843 void openlog(const char *, int, int);
13844 int setlogmask(int);
13845 void syslog(int, const char *, ...);
13846 **APPLICATION USAGE**
13847 None.
13848 **RATIONALE**
13849 None.
13850 **FUTURE DIRECTIONS**
13851 None.
13852 **SEE ALSO**
13853 The System Interfaces volume of IEEE Std. 1003.1-200x, *closelog()*
13854 **CHANGE HISTORY**
13855 First released in Issue 4, Version 2.
13856 **Issue 5**
13857 Moved to X/Open UNIX to BASE.

13858 **NAME**

13859 tar.h — extended tar definitions

13860 **SYNOPSIS**

13861 #include <tar.h>

13862 **DESCRIPTION**

13863 The <tar.h> header shall define header block definitions as follows.

13864 General definitions:

13865

13866

13867

13868

13869

13870

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.

13871

Typeflag field definitions:

13872

13873

13874

13875

13876

13877

13878

13879

13880

13881

13882

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.

13883

Mode field bit definitions (octal):

13884

13885

13886

13887

13888 XSI

13889

13890

13891

13892

13893

13894

13895

13896

13897

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.

13898 **APPLICATION USAGE**

13899 None.

13900 **RATIONALE**

13901 None.

13902 **FUTURE DIRECTIONS**

13903 None.

13904 **SEE ALSO**13905 The Shell and Utilities volume of IEEE Std. 1003.1-200x, *pax*13906 **CHANGE HISTORY**

13907 First released in Issue 3. Derived from the entry in the POSIX.1-1988 standard.

13908 **Issue 4**

13909 This entry is moved from the Headers Interface, Issue 3 specification.

13910 **Issue 4, Version 2**

13911 The following changes are incorporated for X/OPEN UNIX conformance:

13912 • The significance of SYMTYPE as the value of the *typeflag* field is explained.13913 • The value of TSVTX as the value of the *mode* field is explained.13914 **Issue 6**13915 The SEE ALSO section now refers to *pax* since the Shell and Utilities volume of
13916 IEEE Std. 1003.1-200x no longer contains the *tar* utility.

13917 **NAME**

13918 termios.h — define values for termios

13919 **SYNOPSIS**

13920 #include <termios.h>

13921 **DESCRIPTION**

13922 The <termios.h> header contains the definitions used by the terminal I/O interfaces (see
13923 Chapter 11 (on page 213) for the structures and names defined).

13924 **The termios Structure**

13925 The following data types shall be defined through **typedef**:

13926 **cc_t** Used for terminal special characters.

13927 **speed_t** Used for terminal baud rates.

13928 **tcflag_t** Used for terminal modes.

13929 The above types shall be all unsigned integer types.

13930 The **termios** structure shall be defined, and shall include at least the following members:

13931 tcflag_t c_iflag Input modes.
13932 tcflag_t c_oflag Output modes.
13933 tcflag_t c_cflag Control modes.
13934 tcflag_t c_lflag Local modes.
13935 cc_t c_cc[NCCS] Control characters.

13936 A definition shall be provided for:

13937 NCCS Size of the array *c_cc* for control characters.

13938 The following subscript names for the array *c_cc* shall be defined:

13939

13940

13941

13942

13943

13944

13945

13946

13947

13948

13949

13950

13951

13952

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.

13953 The subscript values shall be unique, except that the VMIN and VTIME subscripts may have the
13954 same values as the VEOF and VEOL subscripts, respectively.

13955 The following flags shall be provided.

13956 **Input Modes**

13957 The *c_iflag* field describes the basic terminal input control:

- 13958 BRKINT Signal interrupt on break.
- 13959 ICRNL Map CR to NL on input.
- 13960 IGNBRK Ignore break condition.
- 13961 IGNCR Ignore CR.
- 13962 IGNPAR Ignore characters with parity errors.
- 13963 INLCR Map NL to CR on input.
- 13964 INPCK Enable input parity check.
- 13965 ISTRIP Strip character.
- 13966 XSI IXANY Enable any character to restart output.
- 13967 IXOFF Enable start/stop input control.
- 13968 IXON Enable start/stop output control.
- 13969 PARMRK Mark parity errors.

13970 **Output Modes**

13971 The *c_oflag* field specifies the system treatment of output:

- 13972 OPOST Post-process output.
- 13973 XSI ONLCR Map NL to CR-NL on output.
- 13974 OCRNL Map CR to NL on output.
- 13975 ONOCR No CR output at column 0.
- 13976 ONLRET NL performs CR function.
- 13977 OFILL Use fill characters for delay.
- 13978 NLDLY Select newline delays:
- 13979 NL0 <newline> character type 0.
- 13980 NL1 <newline> character type 1.
- 13981 CRDLY Select carriage-return delays:
- 13982 CR0 Carriage-return delay type 0.
- 13983 CR1 Carriage-return delay type 1.
- 13984 CR2 Carriage-return delay type 2.
- 13985 CR3 Carriage-return delay type 3.
- 13986 TABDLY Select horizontal-tab delays:
- 13987 TAB0 Horizontal-tab delay type 0.
- 13988 TAB1 Horizontal-tab delay type 1.
- 13989 TAB2 Horizontal-tab delay type 2.

13990	TAB3	Expand tabs to spaces.
13991	BSDLY	Select backspace delays:
13992	BS0	Backspace-delay type 0.
13993	BS1	Backspace-delay type 1.
13994	VTDLY	Select vertical-tab delays:
13995	VT0	Vertical-tab delay type 0.
13996	VT1	Vertical-tab delay type 1.
13997	FFDLY	Select form-feed delays:
13998	FF0	Form-feed delay type 0.
13999	FF1	Form-feed delay type 1.

14000 **Baud Rate Selection**

14001 The input and output baud rates are stored in the **termios** structure. These are the valid values
 14002 for objects of type **speed_t**. The following values shall be defined, but not all baud rates need be
 14003 supported by the underlying hardware.

14004	B0	Hang up
14005	B50	50 baud
14006	B75	75 baud
14007	B110	110 baud
14008	B134	134.5 baud
14009	B150	150 baud
14010	B200	200 baud
14011	B300	300 baud
14012	B600	600 baud
14013	B1200	1200 baud
14014	B1800	1800 baud
14015	B2400	2400 baud
14016	B4800	4800 baud
14017	B9600	9600 baud
14018	B19200	19200 baud
14019	B38400	38400 baud

14020 Control Modes

14021 The *c_cflag* field describes the hardware control of the terminal; not all values specified are
14022 required to be supported by the underlying hardware:

14023	CSIZE	Character size:
14024		CS5 5 bits
14025		CS6 6 bits
14026		CS7 7 bits
14027		CS8 8 bits
14028	CSTOPB	Send two stop bits, else one.
14029	CREAD	Enable receiver.
14030	PARENB	Parity enable.
14031	PARODD	Odd parity, else even.
14032	HUPCL	Hang up on last close.
14033	CLOCAL	Ignore modem status lines.

14034 Local Modes

14035 The *c_lflag* field of the argument structure is used to control various terminal functions:

14036	ECHO	Enable echo.
14037	ECHOE	Echo erase character as error-correcting backspace.
14038	ECHOK	Echo KILL.
14039	ECHONL	Echo NL.
14040	ICANON	Canonical input (erase and kill processing).
14041	IEXTEN	Enable extended input character processing.
14042	ISIG	Enable signals.
14043	NOFLSH	Disable flush after interrupt or quit.
14044	TOSTOP	Send SIGTTOU for background output.

14045 Attribute Selection

14046 The following symbolic constants for use with *tcsetattr()* are defined:

14047	TCSANOW	Change attributes immediately.
14048	TCSADRAIN	Change attributes when output has drained.
14049	TCSAFLUSH	Change attributes when output has drained; also flush pending input.

14050 **Line Control**

14051 The following symbolic constants for use with *tflush()* shall be defined:

- 14052 TCIFLUSH Flush pending input. Flush untransmitted output.
- 14053 TCIOFLUSH Flush both pending input and untransmitted output.
- 14054 TCOFLUSH Flush untransmitted output.

14055 The following symbolic constants for use with *tflow()* shall be defined:

- 14056 TCIOFF Transmit a STOP character, intended to suspend input data.
- 14057 TCION Transmit a START character, intended to restart input data.
- 14058 TCOOFF Suspend output.
- 14059 TCOON Restart output.

14060 The following shall be declared as functions and may also be defined as macros. Function
14061 prototypes shall be provided for use with an ISO C standard compiler.

```

14062 speed_t cfgetispeed(const struct termios *);
14063 speed_t cfgetospeed(const struct termios *);
14064 int cfsetispeed(struct termios *, speed_t);
14065 int cfsetospeed(struct termios *, speed_t);
14066 int tcdrain(int);
14067 int tcflow(int, int);
14068 int tcf flush(int, int);
14069 int tcgetattr(int, struct termios *);
14070 xSI pid_t tcgetsid(int);
14071 int tcsendbreak(int, int);
14072 int tcsetattr(int, int, struct termios *);
    
```

14073 **APPLICATION USAGE**

14074 The following names are commonly used as extensions to the above, therefore portable
14075 applications must not use them:

14076 xSI	CBAUD	EXTB	VDSUSP
14077	DEFECHO	FLUSHO	VLNEXT
14078	ECHOCTL	LOBLK	VREPRINT
14079	ECHOKE	PENDIN	VSTATUS
14080	ECHOPRT	SWTCH	VWERASE
14081	EXTA	VDISCARD	

14082 **Note:** These names are not used in IEEE Std. 1003.1-200x, but are reserved for historical use.

14083 **RATIONALE**

14084 None.

14085 **FUTURE DIRECTIONS**

14086 None.

14087 **SEE ALSO**

14088 The System Interfaces volume of IEEE Std. 1003.1-200x, *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*,
14089 *cfsetospeed()*, *tcdrain()*, *tcflow()*, *tcf flush()*, *tcgetattr()*, *tcgetsid()*, *tcsendbreak()*, *tcsetattr()*, Chapter
14090 11 (on page 213)

14091 **CHANGE HISTORY**

14092 First released in Issue 3.

14093 Entry included for alignment with the ISO POSIX-1 standard.

14094 **Issue 4**

14095 The following words are removed from the description of the `c_cc` array: “Implementations that
14096 do not support the job control option, may ignore the SUSP character value in the `c_cc` array
14097 indexed by the VSUSP subscript.” This is because job control is defined as mandatory for Issue 4
14098 conforming implementations.

14099 The mask name symbols IUCLC and OLCUC are marked LEGACY.

14100 The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- 14101 • The function declarations in this header are expanded to full ISO C standard prototypes.
- 14102 • Some minor rewording of the DESCRIPTION is done to align the text more exactly with the
14103 ISO POSIX-1 standard. No functional differences are implied by these changes.
- 14104 • The list of mask name symbols for the `c_oflag` field have all been marked as extensions, with
14105 the exception of OPOST.

14106 **Issue 4, Version 2**

14107 For X/OPEN UNIX conformance, the `tcgetsid()` function is added to the list of functions declared
14108 in this header.

14109 **Issue 6**

14110 The LEGACY symbols IUCLC, ULCUC, and XCASE are removed.

14111 NAME

14112 tgmath.h — type-generic macros

14113 SYNOPSIS

14114 #include <tgmath.h>

14115 DESCRIPTION

14116 cx The functionality described on this reference page extends the ISO C standard. Applications
 14117 shall define the appropriate feature test macro (see the System Interfaces volume of
 14118 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 14119 symbols in this header.

14120 The <tgmath.h> header shall include the headers <math.h> and <complex.h> and shall define
 14121 several type-generic macros.

14122 Of the functions contained within the <math.h> and <complex.h> headers without an *f* (**float**) or
 14123 *l* (**long double**) suffix, several have one or more parameters whose corresponding real type is
 14124 **double**. For each such function, except *modf()*, there shall be a corresponding type-generic
 14125 macro. The parameters whose corresponding real type is **double** in the function synopsis are
 14126 generic parameters. Use of the macro invokes a function whose corresponding real type and
 14127 type domain are determined by the arguments for the generic parameters.

14128 Use of the macro invokes a function whose generic parameters have the corresponding real type
 14129 determined as follows:

- 14130 • First, if any argument for generic parameters has type **long double**, the type determined is
 14131 **long double**.
- 14132 • Otherwise, if any argument for generic parameters has type **double** or is of integer type, the
 14133 type determined is **double**.
- 14134 • Otherwise, the type determined is **float**.

14135 For each unsuffixed function in the <math.h> header for which there is a function in the
 14136 <complex.h> header with the same name except for a *c* prefix, the corresponding type-generic
 14137 macro (for both functions) has the same name as the function in the <math.h> header. The
 14138 corresponding type-generic macro for *fabs()* and *cabs()* is *fabs()*.

14139
14140
14141
14142
14143
14144
14145
14146
14147
14148
14149
14150
14151
14152
14153
14154
14155
14156
14157
14158

<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>

14159 If at least one argument for a generic parameter is complex, then use of the macro invokes a
14160 complex function; otherwise, use of the macro invokes a real function.

14161 For each unsuffixed function in the <math.h> header without a c-prefixed counterpart in the
14162 <complex.h> header, the corresponding type-generic macro has the same name as the function.
14163 These type-generic macros are:

14164	<i>atan2()</i>	<i>fma()</i>	<i>llround()</i>	<i>remainder()</i>
14165	<i>cbrt()</i>	<i>fmax()</i>	<i>log10()</i>	<i>remquo()</i>
14166	<i>ceil()</i>	<i>fmin()</i>	<i>log1p()</i>	<i>rint()</i>
14167	<i>copysign()</i>	<i>fmod()</i>	<i>log2()</i>	<i>round()</i>
14168	<i>erf()</i>	<i>frexp()</i>	<i>logb()</i>	<i>scalbn()</i>
14169	<i>erfc()</i>	<i>hypot()</i>	<i>lrint()</i>	<i>scalbln()</i>
14170	<i>exp2()</i>	<i>ilogb()</i>	<i>lround()</i>	<i>tgamma()</i>
14171	<i>expm1()</i>	<i>ldexp()</i>	<i>nearbyint()</i>	<i>trunc()</i>
14172	<i>fdim()</i>	<i>lgamma()</i>	<i>nextafter()</i>	
14173	<i>floor()</i>	<i>llrint()</i>	<i>nexttoward()</i>	

14174 If all arguments for generic parameters are real, then use of the macro invokes a real function;
14175 otherwise, use of the macro results in undefined behavior.

14176 For each unsuffixed function in the <complex.h> header that is not a c-prefixed counterpart to a
14177 function in the <math.h> header, the corresponding type-generic macro has the same name as
14178 the function. These type-generic macros are:

14179	<i>carg()</i>
14180	<i>cimag()</i>
14181	<i>conj()</i>
14182	<i>cproj()</i>
14183	<i>creal()</i>

14184 Use of the macro with any real or complex argument invokes a complex function.

14185 **APPLICATION USAGE**

14186 With the declarations:

```

14187 #include <tgmath.h>
14188 int n;
14189 float f;
14190 double d;
14191 long double ld;
14192 float complex fc;
14193 double complex dc;
14194 long double complex ldc;
    
```

14195 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>acoshf(f)</i>
<i>sin(d)</i>	<i>sin(d)</i> , the function
<i>atan(ld)</i>	<i>atanl(ld)</i>
<i>log(fc)</i>	<i>clogf(fc)</i>
<i>sqrt(dc)</i>	<i>csqrt(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>

14218 **RATIONALE**

14219 Type-generic macros allow calling a function whose type is determined by the argument type, as
 14220 is the case for C operators such as '+' and '*'. For example, with a type-generic *cos()* macro,
 14221 the expression *cos((float)x)* will have type **float**. This feature enables writing more portably
 14222 efficient code and alleviates need for awkward casting and suffixing in the process of porting or
 14223 adjusting precision. Generic math functions are a widely appreciated feature of Fortran.

14224 The only arguments that affect the type resolution are the arguments corresponding to the
 14225 parameters that have type **double** in the synopsis. Hence the type of a type-generic call to
 14226 *nexttoward()*, whose second parameter is **long double** in the synopsis, is determined solely by
 14227 the type of the first argument.

14228 The term “type-generic” was chosen over the proposed alternatives of intrinsic and overloading.
 14229 The term is more specific than intrinsic, which already is widely used with a more general
 14230 meaning, and reflects a closer match to Fortran’s generic functions than to C++ overloading.

14231 The macros are placed in their own header in order not to silently break old programs that
 14232 include the <math.h> header; for example, with:

14233 `printf ("%e", sin(x))`

14234 *modf(double, double*)* is excluded because no way was seen to make it safe without
14235 complicating the type resolution.

14236 The implementation might, as an extension, endow appropriate ones of the macros that
14237 IEEE Std. 1003.1-200x specifies only for real arguments with the ability to invoke the complex
14238 functions.

14239 IEEE Std. 1003.1-200x does not prescribe any particular implementation mechanism for generic
14240 macros. It could be implemented simply with built-in macros. The generic macro for *sqrt()*, for
14241 example, could be implemented with:

14242 `#undef sqrt`

14243 `#define sqrt(x) __BUILTIN_GENERIC_sqrt(x)`

14244 Generic macros are designed for a useful level of consistency with C++ overloaded math
14245 functions.

14246 The great majority of existing C programs are expected to be unaffected when the **<tgmath.h>**
14247 header is included instead of the **<math.h>** or **<complex.h>** headers. Generic macros are similar
14248 to the ISO/IEC 9899:1999 standard library masking macros, though the semantic types of return
14249 values differ.

14250 The ability to overload on integer as well as floating types would have been useful for some
14251 functions; for example, *copysign()*. Overloading with different numbers of arguments would
14252 have allowed reusing names; for example, *remainder()* for *remquo()*. However, these facilities
14253 would have complicated the specification; and their natural consistent use, such as for a floating
14254 *abs()* or a two-argument *atan()*, would have introduced further inconsistencies with the
14255 ISO/IEC 9899:1999 standard for insufficient benefit.

14256 The ISO C standard in no way limits the implementation's options for efficiency, including
14257 inlining library functions.

14258 **FUTURE DIRECTIONS**

14259 None.

14260 **SEE ALSO**

14261 **<math.h>**, **<complex.h>**, the System Interfaces volume of IEEE Std. 1003.1-200x, *cabs()*, *fabs()*,
14262 *modf()*

14263 **CHANGE HISTORY**

14264 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

14265 **NAME**

14266 time.h — time types

14267 **SYNOPSIS**

14268 #include <time.h>

14269 **DESCRIPTION**

14270 CX The functionality described on this reference page extends the ISO C standard. Applications
 14271 shall define the appropriate feature test macro (see the System Interfaces volume of
 14272 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 14273 symbols in this header.

14274 The <time.h> header shall declare the structure **tm**, which shall include at least the following
 14275 members:

14276	int	tm_sec	Seconds [0,60].
14277	int	tm_min	Minutes [0,59].
14278	int	tm_hour	Hour [0,23].
14279	int	tm_mday	Day of month [1,31].
14280	int	tm_mon	Month of year [0,11].
14281	int	tm_year	Years since 1900.
14282	int	tm_wday	Day of week [0,6] (Sunday =0).
14283	int	tm_yday	Day of year [0,365].
14284	int	tm_isdst	Daylight savings flag.

14285 The value of *tm_isdst* shall be positive if Daylight Saving Time is in effect, 0 if Daylight Saving
 14286 Time is not in effect, and negative if the information is not available.

14287 The <time.h> header shall define the following symbolic names:

14288	NULL	Null pointer constant.
14289	CLOCKS_PER_SEC	A number used to convert the value returned by the <i>clock()</i> function into 14290 seconds.

14291	TMR CPT	CLOCK_PROCESS_CPUTIME_ID
14292		The identifier of the CPU-time clock associated with the process making a 14293 <i>clock()</i> or <i>timer*()</i> function call.

14294	TMR TCT	CLOCK_THREAD_CPUTIME_ID
14295		The identifier of the CPU-time clock associated with the thread making a 14296 <i>clock()</i> or <i>timer*()</i> function call.

14297 TMR The <time.h> header shall declare the structure **timespec**, which has at least the following
 14298 members:

14299	time_t	tv_sec	Seconds.
14300	long	tv_nsec	Nanoseconds.

14301 The <time.h> header shall also declare the **itimerspec** structure, which has at least the following
 14302 members:

14303	struct timespec	it_interval	Timer period.
14304	struct timespec	it_value	Timer expiration.

14305 The following manifest constants shall be defined:

14306	CLOCK_REALTIME	The identifier of the system-wide realtime clock.
14307	TIMER_ABSTIME	Flag indicating time is absolute with respect to the clock associated with a 14308 timer.

14309 MON **CLOCK_MONOTONIC**

14310 The identifier for the system-wide monotonic clock, which is defined as a

14311 clock whose value cannot be set via *clock_settime()* and which cannot

14312 have backward clock jumps. The maximum possible clock jump shall be

14313 implementation-defined.

14314 TMR The **clock_t**, **size_t**, **time_t**, **clockid_t**, and **timer_t** types shall be defined as described in

14315 **<sys/types.h>**.

14316 XSI Although the value of **CLOCKS_PER_SEC** is required to be 1 million on all XSI-conformant

14317 systems, it may be variable on other systems, and it should not be assumed that

14318 **CLOCKS_PER_SEC** is a compile-time constant.

14319 XSI The **<time.h>** header shall provide a declaration for *getdate_err*.

14320 The following shall be declared as functions and may also be defined as macros. Function

14321 prototypes shall be provided for use with an ISO C standard compiler.

14322 char *asctime(const struct tm *);

14323 TSF char *asctime_r(const struct tm *restrict, char *restrict);

14324 clock_t clock(void);

14325 CPT int clock_getcpuclockid(pid_t, clockid_t *);

14326 TMR int clock_getres(clockid_t, struct timespec *);

14327 int clock_gettime(clockid_t, struct timespec *);

14328 CS int clock_nanosleep(clockid_t, int, const struct timespec *,

14329 struct timespec *);

14330 TMR int clock_settime(clockid_t, const struct timespec *);

14331 char *ctime(const time_t *);

14332 TSF char *ctime_r(const time_t *, char *);

14333 double difftime(time_t, time_t);

14334 XSI struct tm *getdate(const char *);

14335 struct tm *gmtime(const time_t *);

14336 struct tm *gmtime_r(const time_t *restrict, struct tm *restrict);

14337 struct tm *localtime(const time_t *);

14338 TSF struct tm *localtime_r(const time_t *restrict, struct tm *restrict);

14339 time_t mktime(struct tm *);

14340 TMR int nanosleep(const struct timespec *, struct timespec *);

14341 size_t strftime(char *restrict, size_t, const char *restrict,

14342 const struct tm *restrict);

14343 XSI char *strptime(const char *restrict, const char *restrict,

14344 struct tm *restrict);

14345 time_t time(time_t *);

14346 TMR int timer_create(clockid_t, struct sigevent *restrict,

14347 timer_t *restrict);

14348 int timer_delete(timer_t);

14349 int timer_gettime(timer_t, struct itimerspec *);

14350 int timer_getoverrun(timer_t);

14351 int timer_settime(timer_t, int, const struct itimerspec *restrict,

14352 struct itimerspec *restrict);

14353 void tzset(void);

14354 The following shall be declared as variables:

14355 XSI extern int daylight;

14356 extern long timezone;

14357 extern char *tzname[];

14358 **APPLICATION USAGE**

14359 The range [0,61] for *tm_sec* allows for the occasional leap second or double leap second.

14360 *tm_year* is a signed value; therefore, years before 1900 may be represented.

14361 To obtain the number of clock ticks per second returned by the *times()* function, applications
14362 should call *sysconf(_SC_CLK_TCK)*.

14363 **RATIONALE**

14364 None.

14365 **FUTURE DIRECTIONS**

14366 None.

14367 **SEE ALSO**

14368 <**sys/types.h**>, the System Interfaces volume of IEEE Std. 1003.1-200x, *asctime()*, *clock()*,
14369 *clock_getcpuclockid()*, *clock_getres()*, *clock_nanosleep()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*,
14370 *localtime()*, *mktime()*, *nanosleep()*, *strftime()*, *strptime()*, *sysconf()*, *time()*, *timer_create()*,
14371 *timer_delete()*, *timer_getoverrun()*, *tzname()*, *tzset()*, *utime()*, the Shell and Utilities volume of
14372 IEEE Std. 1003.1-200x, *daylight*, *timezone*

14373 **CHANGE HISTORY**

14374 First released in Issue 1. Derived from Issue 1 of the SVID.

14375 **Issue 4**

14376 The symbolic name *CLK_TCK* is marked as an extension and LEGACY. Warnings about its use
14377 are also added to the DESCRIPTION.

14378 Reference to the <**sys/types.h**> header is added for the definitions of **clock_t**, **size_t**, and **time_t**.

14379 References to *CLK_TCK* are changed to *CLOCKS_PER_SEC* in part of the DESCRIPTION. The
14380 fact that *CLOCKS_PER_SEC* is always one millionth of a second on XSI-conformant systems is
14381 also marked as an extension.

14382 External declarations for *daylight*, *timezone*, and *tzname* are added. The first two are marked as
14383 extensions.

14384 The *strptime()* function is added to the list of functions declared in this header.

14385 A note about the settings of *tm_sec* is added to the APPLICATION USAGE section.

14386 The following changes are incorporated for alignment with the ISO C standard:

- 14387 • The function declarations in this header are expanded to full ISO C standard prototypes.
- 14388 • The range of *tm_min* is changed from [0,61] to [0,59].
- 14389 • Possible settings of *tm_isdst* and their meanings are added.
- 14390 • The *clock()* and *difftime()* functions are added to the list of functions declared in this header.

14391 **Issue 4, Version 2**

14392 The following changes are incorporated for X/OPEN UNIX conformance:

- 14393 • The <**time.h**> header provides a declaration for *getdate_err*.
- 14394 • The *getdate()* function is added to the list of functions declared in this header.

14395 **Issue 5**

14396 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
14397 Threads Extension.

14398 **Issue 6**

14399 The Open Group corrigenda item U035/6 has been applied. In the DESCRIPTION, the types
14400 **clockid_t** and **timer_t** have been described.

14401 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 14402 • The POSIX timer-related functions are now marked as part of the Timers option.

14403 The symbolic name CLK_TCK is removed. Application usage is added describing how its
14404 equivalent functionality can be obtained using *sysconf()*.

14405 The *clock_getcpuclockid()* function and manifest constants CLOCK_PROCESS_CPUTIME_ID and
14406 CLOCK_THREAD_CPUTIME_ID are added for alignment with IEEE Std. 1003.1d-1999.

14407 The manifest constant CLOCK_MONOTONIC and the *clock_nanosleep()* function are added for
14408 alignment with IEEE Std. 1003.1j-2000.

14409 The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

- 14410 • The range for seconds is changed from 0,61 to 0.60.
- 14411 • The **restrict** keyword is added to the prototypes for *asctime_r()*, *gmtime_r()*, *localtime_r()*,
14412 *strftime()*, *strptime()*, *timer_create()*, and *timer_settime()*.

14413 **NAME**

14414 trace.h — tracing

14415 **SYNOPSIS**

14416 TRC #include <tracing.h>

14417

14418 **DESCRIPTION**

14419 The <trace.h> header shall define the **posix_trace_event_info** structure that includes at least the
 14420 following members:

14421	trace_event_id_t	posix_event_id
14422	pid_t	posix_pid
14423	void	*posix_prog_address
14424	int	posix_truncation_status
14425	struct timespec	posix_timestamp
14426 THR	pthread_t	posix_thread_id

14427

14428 The <trace.h> header shall define the **posix_trace_status_info** structure that includes at least the
 14429 following members:

14430	int	posix_stream_status
14431	int	posix_stream_full_status
14432	int	posix_stream_overrun_status
14433 TRL	int	posix_stream_flush_status
14434	int	posix_stream_flush_error
14435	int	posix_log_overrun_status
14436	int	posix_log_full_status

14437

14438 The <trace.h> header shall define the following symbols:

14439	POSIX_TRACE_RUNNING
14440	POSIX_TRACE_SUSPENDED
14441	POSIX_TRACE_FULL
14442	POSIX_TRACE_NOT_FULL
14443	POSIX_TRACE_NO_OVERRUN
14444	POSIX_TRACE_OVERRUN
14445 TRL	POSIX_TRACE_FLUSHING
14446	POSIX_TRACE_NOT_FLUSHING
14447	POSIX_TRACE_NOT_TRUNCATED
14448	POSIX_TRACE_TRUNCATED_READ
14449	POSIX_TRACE_TRUNCATED_RECORD
14450 TRL	POSIX_TRACE_FLUSH
14451	POSIX_TRACE_LOOP
14452	POSIX_TRACE_UNTIL_FULL
14453 TRI	POSIX_TRACE_CLOSE_FOR_CHILD
14454	POSIX_TRACE_INHERITED
14455 TRL	POSIX_TRACE_APPEND
14456	POSIX_TRACE_LOOP
14457	POSIX_TRACE_UNTIL_FULL
14458 TEF	POSIX_TRACE_FILTER
14459 TRL	POSIX_TRACE_FLUSH_START
14460	POSIX_TRACE_FLUSH_STOP
14461	POSIX_TRACE_OVERFLOW

14462 POSIX_TRACE_RESUME
 14463 POSIX_TRACE_START
 14464 POSIX_TRACE_STOP
 14465 POSIX_TRACE_UNNAMED_USER_EVENT

14466 The following types shall be defined as described in **<sys/types.h>**:

14467 **trace_attr_t**
 14468 **trace_id_t**
 14469 **trace_event_id_t**
 14470 TEF **trace_event_set_t**
 14471

14472 The following shall be declared as functions and may also be declared as macros. Function
 14473 prototypes shall be provided for use with an ISO C standard compiler.

```

14474 int posix_trace_attr_destroy(trace_attr_t *);
14475 int posix_trace_attr_getclockres(const trace_attr_t *,
14476     struct timespec *);
14477 int posix_trace_attr_getcreatetime(const trace_attr_t *,
14478     struct timespec *);
14479 int posix_trace_attr_getgenversion(const trace_attr_t *, char *);
14480 TRI int posix_trace_attr_getinherited(const trace_attr_t *, int *);
14481 TRL int posix_trace_attr_getlogfullpolicy(const trace_attr_t *, int *);
14482 int posix_trace_attr_getlogsize(const trace_attr_t *, size_t *);
14483 int posix_trace_attr_getmaxdatasize(const trace_attr_t *, size_t *);
14484 int posix_trace_attr_getmaxsystemevents(size_t *,
14485     size_t *);
14486 int posix_trace_attr_getmaxuserevents(size_t *,
14487     size_t *);
14488 int posix_trace_attr_getname(const trace_attr_t *, char *);
14489 int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *, int *);
14490 int posix_trace_attr_getstreamsize(const trace_attr_t *, size_t *);
14491 int posix_trace_attr_init(trace_attr_t *);
14492 TRI int posix_trace_attr_setinherited(trace_attr_t *, int);
14493 TRL int posix_trace_attr_setlogfullpolicy(trace_attr_t *, int);
14494 int posix_trace_attr_setlogsize(trace_attr_t *, size_t);
14495 int posix_trace_attr_setmaxdatasize(trace_attr_t *, size_t);
14496 int posix_trace_attr_setname(trace_attr_t *, const char *);
14497 int posix_trace_attr_setstreamsize(trace_attr_t *, size_t);
14498 int posix_trace_attr_setstreamfullpolicy(trace_attr_t *, int);
14499 int posix_trace_clear(trace_id_t);
14500 TRL int posix_trace_close(trace_id_t);
14501 int posix_trace_create(pid_t, const trace_attr_t *, trace_id_t *);
14502 TRL int posix_trace_create_withlog(pid_t, const trace_attr_t *, int,
14503     trace_id_t *);
14504 void posix_trace_event(trace_event_id_t, const void *, size_t);
14505 int posix_trace_eventid_equal(trace_id_t, trace_eventid_t,
14506     trace_eventid_t);
14507 int posix_trace_eventid_get_name(trace_id_t, trace_eventid_t, char *);
14508 int posix_trace_eventid_open(const char *, trace_event_id_t *);
14509 int posix_trace_eventtypelist_getnext_id(trace_id_t, trace_eventid_t *,
14510     int *);
14511 int posix_trace_eventtypelist_rewind(trace_id_t);

```



```

14512 TEF      int  posix_trace_eventset_add(trace_event_id_t, trace_event_set_t *);
14513         int  posix_trace_eventset_del(trace_event_id_t, trace_event_set_t *);
14514         int  posix_trace_eventset_empty(trace_event_set_t *);
14515         int  posix_trace_eventset_fill(trace_event_set_t *, int);
14516         int  posix_trace_eventset_ismember(trace_event_id_t,
14517             const trace_event_set_t *, int *);
14518         int  posix_trace_flush(trace_id_t);
14519         int  posix_trace_get_attr(trace_id_t, trace_attr_t *);
14520 TEF      int  posix_trace_get_filter(trace_id_t, trace_event_set_t *);
14521         int  posix_trace_get_status(trace_id_t,
14522             struct posix_trace_status_info *);
14523         int  posix_trace_getnext_event(trace_id_t,
14524             struct posix_trace_event_info *, void *, size_t, size_t *,
14525             int *);
14526 TRL      int  posix_trace_open(int, trace_id_t *);
14527         int  posix_trace_rewind(trace_id_t);
14528 TEF      int  posix_trace_set_filter(trace_id_t, const trace_event_set_t *, int);
14529         int  posix_trace_shutdown(trace_id_t);
14530         int  posix_trace_start(trace_id_t);
14531         int  posix_trace_stop(trace_id_t);
14532 TMO      int  posix_trace_timedgetnext_event(trace_id_t,
14533             struct posix_trace_event_info *, void *, size_t, size_t *,
14534             int *, const struct timespec *);
14535 TEF      int  posix_trace_trid_eventid_open(trace_id_t, const char *,
14536             trace_eventid_t *);
14537         int  posix_trace_trygetnext_event(trace_id_t,
14538             struct posix_trace_event_info *, void *, size_t, size_t *,
14539             int *);

```

14540 **APPLICATION USAGE**

14541 None.

14542 **RATIONALE**

14543 None.

14544 **FUTURE DIRECTIONS**

14545 None.

14546 **SEE ALSO**

14547 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, Section 2.11, Tracing, the
14548 System Interfaces volume of IEEE Std. 1003.1-200x, *posix_trace_attr_destroy()*,
14549 *posix_trace_attr_getclockres()*, *posix_trace_attr_getcreatetime()*, *posix_trace_attr_getgenversion()*,
14550 *posix_trace_attr_getinherited()*, *posix_trace_attr_getlogfullpolicy()*, *posix_trace_attr_getlogsize()*,
14551 *posix_trace_attr_getmaxdatasize()*, *posix_trace_attr_getmaxsystemeventsizesize()*,
14552 *posix_trace_attr_getmaxusereventsizesize()*, *posix_trace_attr_getname()*,
14553 *posix_trace_attr_getstreamfullpolicy()*, *posix_trace_attr_getstreamsize()*, *posix_trace_attr_init()*,
14554 *posix_trace_attr_setinherited()*, *posix_trace_attr_setlogfullpolicy()*, *posix_trace_attr_setlogsize()*,
14555 *posix_trace_attr_setmaxdatasize()*, *posix_trace_attr_setname()*, *posix_trace_attr_setstreamsize()*,
14556 *posix_trace_attr_setstreamfullpolicy()*, *posix_trace_clear()*, *posix_trace_close()*, *posix_trace_create()*,
14557 *posix_trace_create_withlog()*, *posix_trace_event()*, *posix_trace_eventid_equal()*,
14558 *posix_trace_eventid_get_name()*, *posix_trace_eventid_open()*, *posix_trace_eventtypelist_getnext_id()*,
14559 *posix_trace_eventtypelist_rewind()*, *posix_trace_eventset_add()*, *posix_trace_eventset_del()*,
14560 *posix_trace_eventset_empty()*, *posix_trace_eventset_fill()*, *posix_trace_eventset_ismember()*,
14561 *posix_trace_flush()*, *posix_trace_get_attr()*, *posix_trace_get_filter()*, *posix_trace_get_status()*,

14562 *posix_trace_getnext_event(), posix_trace_open(), posix_trace_rewind(), posix_trace_set_filter(),*
14563 *posix_trace_shutdown(), posix_trace_start(), posix_trace_stop(), posix_trace_timedgetnext_event(),*
14564 *posix_trace_trid_eventid_open(), posix_trace_trygetnext_event()*

14565 **CHANGE HISTORY**

14566 First released in Issue 6. Derived from IEEE Std. 1003.1q-2000.

14567 **NAME**

14568 ucontext.h — user context

14569 **SYNOPSIS**

14570 XSI `#include <ucontext.h>`

14571

14572 **DESCRIPTION**

14573 The <ucontext.h> header shall define the **mcontext_t** type through **typedef**.

14574 The <ucontext.h> header shall define the **ucontext_t** type as a structure that shall include at least
14575 the following members:

14576	ucontext_t *uc_link	Pointer to the context that is resumed when this context returns.
14577		
14578	sigset_t uc_sigmask	The set of signals that are blocked when this context is active.
14579		
14580	stack_t uc_stack	The stack used by this context.
14581	mcontext_t uc_mcontext	A machine-specific representation of the saved context.
14582		

14583 The types **sigset_t** and **stack_t** shall be defined as in <signal.h>.

14584 The following shall be declared as functions and may also be defined as macros, Function
14585 prototypes shall be provided for use with an ISO C standard compiler.

```
14586 int getcontext(ucontext_t *);
14587 int setcontext(const ucontext_t *);
14588 void makecontext(ucontext_t *, void (*)(void), int, ...);
14589 int swapcontext(ucontext_t *restrict, const ucontext_t *restrict);
```

14590 **APPLICATION USAGE**

14591 None.

14592 **RATIONALE**

14593 None.

14594 **FUTURE DIRECTIONS**

14595 None.

14596 **SEE ALSO**

14597 <signal.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *getcontext()*, *makecontext()*,
14598 *sigaction()*, *sigprocmask()*, *sigaltstack()*

14599 **CHANGE HISTORY**

14600 First released in Issue 4, Version 2.

14601 **NAME**

14602 ulimit.h — ulimit commands

14603 **SYNOPSIS**

14604 XSI #include <ulimit.h>

14605

14606 **DESCRIPTION**14607 The <ulimit.h> header shall define the symbolic constants used by the *ulimit()* function.

14608 Symbolic constants:

14609 UL_GETFSIZE Get maximum file size.

14610 UL_SETFSIZE Set maximum file size.

14611 The following shall be declared as a function and may also be defined as a macro. Function
14612 prototypes shall be provided for use with an ISO C standard compiler.

14613 long ulimit(int, ...);

14614 **APPLICATION USAGE**

14615 None.

14616 **RATIONALE**

14617 None.

14618 **FUTURE DIRECTIONS**

14619 None.

14620 **SEE ALSO**14621 The System Interfaces volume of IEEE Std. 1003.1-200x, *ulimit()*14622 **CHANGE HISTORY**

14623 First released in Issue 3.

14624 **Issue 4**

14625 The function declarations in this header are expanded to full ISO C standard prototypes.

14626 **NAME**

14627 unistd.h — standard symbolic constants and types

14628 **SYNOPSIS**

14629 #include <unistd.h>

14630 **DESCRIPTION**

14631 The <unistd.h> header defines miscellaneous symbolic constants and types, and declares
 14632 miscellaneous functions. The actual value of the constants are unspecified except as shown. The
 14633 contents of this header are shown below.

14634 **Version Test Macros**

14635 The following symbolic constants shall be defined:

14636 **_POSIX_VERSION**

14637 Integer value indicating version of IEEE Std. 1003.1-200x (C-language binding). The value is
 14638 200xxxL. This value shall be used for systems that conform to IEEE Std. 1003.1-200x.

14639 **_POSIX2_VERSION**

14640 Integer value indicating version of the Shell and Utilities volume of IEEE Std. 1003.1-200x.

14641 XSI **_XOPEN_VERSION**

14642 Integer value indicating version of the X/Open Portability Guide to which the
 14643 implementation conforms. The value is 600.

14644 XSI **_XOPEN_XCU_VERSION** is defined as an integer value indicating the version of the Shell and
 14645 Utilities volume of IEEE Std. 1003.1-200x to which the implementation conforms. If the value is
 14646 -1, no commands and utilities are provided on the implementation. If the value is greater than
 14647 or equal to 4, the functionality associated with the following symbols is also supported (see
 14648 **Constants for Options and Option Groups** (on page 438) and **Constants for Profiling Option**
 14649 **Groups** (on page 444)):

14650 **_POSIX2_C_BIND**

14651 **_POSIX2_CHAR_TERM**

14652 **_POSIX2_LOCALEDEF**

14653 **_POSIX2_UPE**

14654 **_POSIX2_VERSION**

14655 If **_XOPEN_XCU_VERSION** is not defined, use the *sysconf()* function to determine which
 14656 features are supported.

14657 Each of the following symbolic constants shall be defined only if the implementation supports
 14658 the indicated version of the X/Open Portability Guide:

14659 XSI **_XOPEN_UNIX**

14660 X/Open CAE Specification, January 1997, System Interfaces and Headers, Issue 5
 14661 (ISBN: 1-85912-181-0, C606).

14662 **_XOPEN_XPG2**

14663 X/Open Portability Guide, Volume 2, January 1987, XVS System Calls and Libraries
 14664 (ISBN: 0-444-70175-3).

14665 **_XOPEN_XPG3**

14666 X/Open Specification, February 1992, System Interfaces and Headers, Issue 3
 14667 (ISBN: 1-872630-37-5, C212); this specification was formerly X/Open Portability Guide,
 14668 Issue 3, Volume 2, January 1989, XSI System Interface and Headers (ISBN: 0-13-685843-0,
 14669 XO/XPG/89/003).

14670 **_XOPEN_XPG4**
 14671 X/Open CAE Specification, July 1992, System Interfaces and Headers, Issue 4
 14672 (ISBN: 1-872630-47-2, C202).

14673 **Constants for Options and Option Groups**

14674 The following symbolic constants, if defined in <unistd.h>, shall have a value of -1, 0, or greater,
 14675 unless otherwise specified below. If these are undefined, the *sysconf()* function can be used to
 14676 determine whether the option is provided for a particular invocation of the application.

14677 If a symbolic constant is defined with the value -1, the option is not supported. Headers, data
 14678 types, and function interfaces required only for the option need not be supplied. An application
 14679 that attempts to use anything associated only with the option is considered to be requiring an
 14680 extension.

14681 If a symbolic constant is defined with a value greater than zero, the option shall always be
 14682 supported when the application is executed. All headers, data types, and functions shall be
 14683 present and shall operate as specified.

14684 If a symbolic constant is defined with the value zero, all headers, data types, and functions shall
 14685 be present. The application must check at runtime to see whether the option is supported by
 14686 calling *sysconf()* with the indicated *name* parameter.

14687 Unless explicitly specified otherwise, the behavior of functions associated with an unsupported
 14688 option is unspecified, and an application that uses such functions without first checking
 14689 *sysconf()* is considered to be requiring an extension.

14690 For conformance requirements, refer to Chapter 2 (on page 19).

14691 **ADV** **_POSIX_ADVISORY_INFO**
 14692 The implementation supports the Advisory Information option. If this symbol has a value
 14693 other than -1, it shall have the value 200ymmL, the date of approval of
 14694 IEEE Std. 1003.1-200x.

14695 **AIO** **_POSIX_ASYNCHRONOUS_IO**
 14696 The implementation supports the Asynchronous Input and Output option. If this symbol
 14697 has a value other than -1, it shall have the value 200ymmL, the date of approval of
 14698 IEEE Std. 1003.1-200x.

14699 **BAR** **_POSIX_BARRIERS**
 14700 The implementation supports the Barriers option. If this symbol has a value other than -1, it
 14701 shall have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.

14702 **_POSIX_CHOWN_RESTRICTED**
 14703 The use of *chown()* and *fchown()* is restricted to a process with appropriate privileges, and
 14704 to changing the group ID of a file only to the effective group ID of the process or to one of
 14705 its supplementary group IDs.

14706 **CS** **_POSIX_CLOCK_SELECTION**
 14707 The implementation supports the Clock Selection option. If this symbol has a value other
 14708 than -1, it shall have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.

14709 **CPT** **_POSIX_CPUTIME**
 14710 The implementation supports the Process CPU-Time Clocks option. If this symbol has a
 14711 value other than -1, it shall have the value 200ymmL, the date of approval of
 14712 IEEE Std. 1003.1-200x.

14713 **FSC** **_POSIX_FSYNC**
 14714 The implementation supports the File Synchronization option. If this symbol has a value

14715	other than -1, it shall have the value 200ymmL, the date of approval of
14716	IEEE Std. 1003.1-200x.
14717	_POSIX_JOB_CONTROL
14718	The implementation supports job control. This is always set to a value greater than zero.
14719 MF	_POSIX_MAPPED_FILES
14720	The implementation supports the Memory Mapped Files option. If this symbol has a value
14721	other than -1, it shall have the value 200ymmL, the date of approval of
14722	IEEE Std. 1003.1-200x.
14723 ML	_POSIX_MEMLOCK
14724	The implementation supports the Process Memory Locking option. If this symbol has a
14725	value other than -1, it shall have the value 200ymmL, the date of approval of
14726	IEEE Std. 1003.1-200x.
14727 MLR	_POSIX_MEMLOCK_RANGE
14728	The implementation supports the Range Memory Locking option. If this symbol has a value
14729	other than -1, it shall have the value 200ymmL, the date of approval of
14730	IEEE Std. 1003.1-200x.
14731 MPR	_POSIX_MEMORY_PROTECTION
14732	The implementation supports the Memory Protection option. If this symbol has a value
14733	other than -1, it shall have the value 200ymmL, the date of approval of
14734	IEEE Std. 1003.1-200x.
14735 MSG	_POSIX_MESSAGE_PASSING
14736	The implementation supports the Message Passing option. If this symbol has a value other
14737	than -1, it shall have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.
14738 MON	_POSIX_MONOTONIC_CLOCK
14739	The implementation supports the Monotonic Clock option. If this symbol has a value other
14740	than -1, it shall have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.
14741	_POSIX_NO_TRUNC
14742	Path name components longer than {NAME_MAX} generate an error.
14743 PIO	_POSIX_PRIORITIZED_IO
14744	The implementation supports the Prioritized Input and Output option. If this symbol has a
14745	value other than -1, it shall have the value 200ymmL, the date of approval of
14746	IEEE Std. 1003.1-200x.
14747 PS	_POSIX_PRIORITY_SCHEDULING
14748	The implementation supports the Process Scheduling option. If this symbol has a value
14749	other than -1, it shall have the value 200ymmL, the date of approval of
14750	IEEE Std. 1003.1-200x.
14751 THR	_POSIX_READER_WRITER_LOCKS
14752	The implementation supports the Read-Write Locks option. This is always set to a value
14753	greater than zero if the Threads option is supported. If this symbol has a value other than
14754	-1, it shall have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.
14755 RTS	_POSIX_REALTIME_SIGNALS
14756	The implementation supports the Realtime Signals Extension option. If this symbol has a
14757	value other than -1, it shall have the value 200ymmL, the date of approval of
14758	IEEE Std. 1003.1-200x.
14759	_POSIX_REGEX
14760	The implementation supports the Regular Expression Handling option. This is always set

14761	to a value greater than zero.
14762	_POSIX_SAVED_IDS
14763	Each process has a saved set-user-ID and a saved set-group-ID. The behavior of the <i>setuid()</i> ,
14764	<i>setgid()</i> , and <i>kill()</i> functions shall be dependent on the values of the saved set-user-ID and
14765	the saved get-group-ID, respectively. This is always set to a value greater than zero.
14766 SEM	_POSIX_SEMAPHORES
14767	The implementation supports the Semaphores option. If this symbol has a value other than
14768	-1, it shall have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.
14769 SHM	_POSIX_SHARED_MEMORY_OBJECTS
14770	The implementation supports the Shared Memory Objects option. If this symbol has a value
14771	other than -1, it shall have the value 200ymmL, the date of approval of
14772	IEEE Std. 1003.1-200x.
14773 SH	_POSIX_SHELL
14774	The implementation supports the POSIX shell. This is always set to a value greater than
14775	zero.
14776 SPN	_POSIX_SPAWN
14777	The implementation supports the Spawn option. If this symbol has a value other than -1, it
14778	shall have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.
14779 SPI	_POSIX_SPIN_LOCKS
14780	The implementation supports the Spin Locks option. If this symbol has a value other than
14781	-1, it shall have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.
14782 SS	_POSIX_SPORADIC_SERVER
14783	The implementation supports the Process Sporadic Server option. If this symbol has a value
14784	other than -1, it shall have the value 200ymmL, the date of approval of
14785	IEEE Std. 1003.1-200x.
14786 SIO	_POSIX_SYNCHRONIZED_IO
14787	The implementation supports the Synchronized Input and Output option. If this symbol
14788	has a value other than -1, it shall have the value 200ymmL, the date of approval of
14789	IEEE Std. 1003.1-200x.
14790 TSA	_POSIX_THREAD_ATTR_STACKADDR
14791	The implementation supports the Thread Stack Address Attribute option. If this symbol
14792	has a value other than -1, it shall have the value 200ymmL, the date of approval of
14793	IEEE Std. 1003.1-200x.
14794 TSS	_POSIX_THREAD_ATTR_STACKSIZE
14795	The implementation supports the Thread Stack Address Size option. If this symbol has a
14796	value other than -1, it shall have the value 200ymmL, the date of approval of
14797	IEEE Std. 1003.1-200x.
14798 TCT	_POSIX_THREAD_CPUTIME
14799	The implementation supports the Thread CPU-Time Clocks option. If this symbol has a
14800	value other than -1, it shall have the value 200ymmL, the date of approval of
14801	IEEE Std. 1003.1-200x.
14802 TPI	_POSIX_THREAD_PRIO_INHERIT
14803	The implementation supports the Threads Priority Inheritance option. If this symbol has a
14804	value other than -1, it shall have the value 200ymmL, the date of approval of
14805	IEEE Std. 1003.1-200x.

14806	TPP	_POSIX_THREAD_PRIO_PROTECT
14807		The implementation supports the Thread Priority Protection option. If this symbol has a
14808		value other than <code>-1</code> , it shall have the value <code>200ymmL</code> , the date of approval of
14809		IEEE Std. 1003.1-200x.
14810	TPS	_POSIX_THREAD_PRIORITY_SCHEDULING
14811		The implementation supports the Thread Execution Scheduling option. If this symbol has a
14812		value other than <code>-1</code> , it shall have the value <code>200ymmL</code> , the date of approval of
14813		IEEE Std. 1003.1-200x.
14814	TSH	_POSIX_THREAD_PROCESS_SHARED
14815		The implementation supports the Thread Process-Shared Synchronization option. If this
14816		symbol has a value other than <code>-1</code> , it shall have the value <code>200ymmL</code> , the date of approval of
14817		IEEE Std. 1003.1-200x.
14818	TSF	_POSIX_THREAD_SAFE_FUNCTIONS
14819		The implementation supports the Thread-Safe Functions option. If this symbol has a value
14820		other than <code>-1</code> , it shall have the value <code>200ymmL</code> , the date of approval of
14821		IEEE Std. 1003.1-200x.
14822	TSP	_POSIX_THREAD_SPORADIC_SERVER
14823		The implementation supports the Thread Sporadic Server option. If this symbol has a value
14824		other than <code>-1</code> , it shall have the value <code>200ymmL</code> , the date of approval of
14825		IEEE Std. 1003.1-200x.
14826	THR	_POSIX_THREADS
14827		The implementation supports the Threads option. If this symbol has a value other than <code>-1</code> , it
14828		shall have the value <code>200ymmL</code> , the date of approval of IEEE Std. 1003.1-200x.
14829	TMR	_POSIX_TIMERS
14830		The implementation supports the Timers option. If this symbol has a value other than <code>-1</code> , it
14831		shall have the value <code>200ymmL</code> , the date of approval of IEEE Std. 1003.1-200x.
14832	TMO	_POSIX_TIMEOUTS
14833		The implementation supports the Timeouts option. If this symbol has a value other than <code>-1</code> ,
14834		it shall have the value <code>200ymmL</code> , the date of approval of IEEE Std. 1003.1-200x.
14835	TRC	_POSIX_TRACE
14836		The implementation supports the Trace option.
14837	TEF	_POSIX_TRACE_EVENT_FILTER
14838		The implementation supports the Trace Event Filter option.
14839	TRL	_POSIX_TRACE_LOG
14840		The implementation supports the Trace Log option.
14841	TRI	_POSIX_TRACE_INHERIT
14842		The implementation supports the Trace Inherit option.
14843	TYM	_POSIX_TYPED_MEMORY_OBJECTS
14844		The implementation supports the Typed Memory Objects option. If this symbol has a value
14845		other than <code>-1</code> , it shall have the value <code>200ymmL</code> , the date of approval of
14846		IEEE Std. 1003.1-200x.
14847		_POSIX_VDISABLE
14848		Terminal special characters defined in <termios.h> can be disabled using this character
14849		value.

14850 _POSIX2_C_BIND
14851 The implementation supports the C-Language Binding option. This always has the value
14852 200ymmL, the date of approval of IEEE Std. 1003.1-200x.

14853 CD _POSIX2_C_DEV
14854 The implementation supports the C-Language Development Utilities option. If this symbol
14855 has a value other than -1, it shall have the value 200ymmL, the date of approval of
14856 IEEE Std. 1003.1-200x.

14857 _POSIX2_CHAR_TERM
14858 The implementation supports at least one terminal type.

14859 FD _POSIX2_FORT_DEV
14860 The implementation supports the FORTRAN Development Utilities option. If this symbol
14861 has a value other than -1, it shall have the value 200ymmL, the date of approval of
14862 IEEE Std. 1003.1-200x.

14863 FR _POSIX2_FORT_RUN
14864 The implementation supports the FORTRAN Runtime Utilities option. If this symbol has a
14865 value other than -1, it shall have the value 200ymmL, the date of approval of
14866 IEEE Std. 1003.1-200x.

14867 _POSIX2_LOCALEDEF
14868 The implementation supports the creation of locales by the *localedef* utility. If this symbol
14869 has a value other than -1, it shall have the value 200ymmL, the date of approval of
14870 IEEE Std. 1003.1-200x.

14871 BE _POSIX2_PBS
14872 The implementation supports the Batch Environment Services and Utilities option. If this
14873 symbol has a value other than -1, it shall have the value 200ymmL, the date of approval of
14874 IEEE Std. 1003.1-200x.

14875 BE _POSIX2_PBS_ACCOUNTING
14876 The implementation supports the Batch Accounting option. If this symbol has a value other
14877 than -1, it shall have the value 200ymmL, the date of approval of IEEE Std. 1003.1-200x.

14878 BE _POSIX2_PBS_CHECKPOINT
14879 The implementation supports the Batch Checkpoint/Restart option. If this symbol has a
14880 value other than -1, it shall have the value 200ymmL, the date of approval of
14881 IEEE Std. 1003.1-200x.

14882 BE _POSIX2_PBS_LOCATE
14883 The implementation supports the Locate Batch Job Request option. If this symbol has a
14884 value other than -1, it shall have the value 200ymmL, the date of approval of
14885 IEEE Std. 1003.1-200x.

14886 BE _POSIX2_PBS_MESSAGE
14887 The implementation supports the Batch Job Message Request option. If this symbol has a
14888 value other than -1, it shall have the value 200ymmL, the date of approval of
14889 IEEE Std. 1003.1-200x.

14890 BE _POSIX2_PBS_TRACK
14891 The implementation supports the Track Batch Job Request option. If this symbol has a value
14892 other than -1, it shall have the value 200ymmL, the date of approval of
14893 IEEE Std. 1003.1-200x.

14894 SD _POSIX2_SW_DEV
14895 The implementation supports the Software Development Utilities option. If this symbol has

14896	a value other than <code>-1</code> , it shall have the value <code>200ymmL</code> , the date of approval of
14897	IEEE Std. 1003.1-200x.
14898 UP	_POSIX2_UPE
14899	The implementation supports the User Portability Utilities option. If this symbol has a value
14900	other than <code>-1</code> , it shall have the value <code>200ymmL</code> , the date of approval of
14901	IEEE Std. 1003.1-200x.
14902	_V6_ILP32_OFF32
14903	The implementation provides a C-language compilation environment with 32-bit int , long ,
14904	pointer , and off_t types.
14905	_V6_ILP32_OFFBIG
14906	The implementation provides a C-language compilation environment with 32-bit int , long ,
14907	and pointer types and an off_t type using at least 64 bits.
14908	_V6_LP64_OFF64
14909	The implementation provides a C-language compilation environment with 32-bit int and
14910	64-bit long , pointer , and off_t types.
14911	_V6_LPBIG_OFFBIG
14912	The implementation provides a C-language compilation environment with an int type
14913	using at least 32 bits and long , pointer , and off_t types using at least 64 bits.
14914 XSI	_XBS5_ILP32_OFF32 (LEGACY)
14915	The implementation provides a C-language compilation environment with 32-bit int , long ,
14916	pointer , and off_t types.
14917 XSI	_XBS5_ILP32_OFFBIG (LEGACY)
14918	The implementation provides a C-language compilation environment with 32-bit int , long ,
14919	and pointer types and an off_t type using at least 64 bits.
14920 XSI	_XBS5_LP64_OFF64 (LEGACY)
14921	The implementation provides a C-language compilation environment with 32-bit int and
14922	64-bit long , pointer , and off_t types.
14923 XSI	_XBS5_LPBIG_OFFBIG (LEGACY)
14924	The implementation provides a C-language compilation environment with an int type
14925	using at least 32 bits and long , pointer , and off_t types using at least 64 bits.
14926 XSI	_XOPEN_CRYPT
14927	The implementation supports the X/Open Encryption Option Group.
14928	_XOPEN_ENH_I18N
14929	The implementation supports the Issue 4, Version 2 Enhanced Internationalization Option
14930	Group. This is always set to a value other than <code>-1</code> .
14931	_XOPEN_LEGACY
14932	The implementation supports the Legacy Option Group.
14933	_XOPEN_REALTIME
14934	The implementation supports the X/Open Realtime Option Group.
14935	_XOPEN_REALTIME_THREADS
14936	The implementation supports the X/Open Realtime Threads Option Group.
14937	_XOPEN_SHM
14938	The implementation supports the Issue 4, Version 2 Shared Memory Option Group. This is
14939	always set to a value other than <code>-1</code> .

14940 **XOPEN_STREAMS**
14941 The implementation supports the XSI STREAMS Option Group.

14942 **Constants for Profiling Option Groups**

14943 The following symbolic constants shall be defined to have the value -1 if the implementation
14944 never provides the Profiling Option Group, and to have a value other than -1 if the
14945 implementation always provides the Profiling Option Group. If these are undefined, the
14946 *sysconf()* function can be used to determine whether the Profiling Option Group is provided for
14947 a particular invocation of the application.

14948 For conformance requirements, refer to Chapter 2 (on page 19).

- 14949 • `_POSIX_BASE`
- 14950 • `_POSIX_C_LANG_SUPPORT`
- 14951 • `_POSIX_C_LANG_SUPPORT_R`
- 14952 • `_POSIX_DEVICE_IO`
- 14953 • `_POSIX_DEVICE_SPECIFIC`
- 14954 • `_POSIX_DEVICE_SPECIFIC_R`
- 14955 • `_POSIX_FD_MGMT`
- 14956 • `_POSIX_FIFO`
- 14957 • `_POSIX_FILE_ATTRIBUTES`
- 14958 • `_POSIX_FILE_LOCKING`
- 14959 • `_POSIX_FILE_SYSTEM`
- 14960 • `_POSIX_JOB_CONTROL`
- 14961 • `_POSIX_MULTIPLE_PROCESS`
- 14962 • `_POSIX_NETWORKING`
- 14963 • `_POSIX_PIPE`
- 14964 • `_POSIX_SIGNALS`
- 14965 • `_POSIX_SINGLE_PROCESS`
- 14966 • `_POSIX_SYSTEM_DATABASE`
- 14967 • `_POSIX_SYSTEM_DATABASE_R`
- 14968 • `_POSIX_USER_GROUPS`
- 14969 • `_POSIX_USER_GROUPS_R`

14970 **Execution-Time Symbolic Constants**

14971 If any of the following constants are not defined in the <unistd.h> header, the value shall vary
14972 depending on the file to which it is applied.

14973 If any of the following constants are defined to have value -1 in the <unistd.h> header, the
14974 implementation shall not provide the option on any file; if any are defined to have a value other
14975 than -1 in the <unistd.h> header, the implementation shall provide the option on all applicable
14976 files.

14977 All of the following constants, whether defined in <unistd.h> or not, may be queried with
14978 respect to a specific file using the *pathconf()* or *fpathconf()* functions:

14979 `_POSIX_ASYNC_IO`

14980 Asynchronous input or output operations may be performed for the associated file.

14981 `_POSIX_PRIO_IO`

14982 Prioritized input or output operations may be performed for the associated file.

14983 `_POSIX_SYNC_IO`

14984 Synchronized input or output operations may be performed for the associated file.

14985 **Constants for Functions**

14986 The following symbolic constant shall be defined:

14987 `NULL` Null pointer

14988 The following symbolic constants shall be defined for the *access()* function:

14989 `F_OK` Test for existence of file.

14990 `R_OK` Test for read permission.

14991 `W_OK` Test for write permission.

14992 `X_OK` Test for execute (search) permission.

14993 The constants `F_OK`, `R_OK`, `W_OK`, and `X_OK` and the expressions `R_OK | W_OK`, `R_OK | X_OK`,
14994 and `R_OK | W_OK | X_OK` shall all have distinct values.

14995 The following symbolic constants shall be defined for the *confstr()* function:

14996 `_CS_PATH`

14997 This is the value for the *PATH* environment variable that finds all standard utilities.

14998 `_CS_V6_ILP32_OFF32_CFLAGS`

14999 If *sysconf*(`_SC_V6_ILP32_OFF32`) returns `-1`, the meaning of this value is unspecified.
15000 Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to
15001 build an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t**
15002 types.

15003 `_CS_V6_ILP32_OFF32_LDFLAGS`

15004 If *sysconf*(`_SC_V6_ILP32_OFF32`) returns `-1`, the meaning of this value is unspecified.
15005 Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build
15006 an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

15007 `_CS_V6_ILP32_OFF32_LIBS`

15008 If *sysconf*(`_SC_V6_ILP32_OFF32`) returns `-1`, the meaning of this value is unspecified.
15009 Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an
15010 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

15011 `_CS_V6_ILP32_OFF32_LINTFLAGS`

15012 If *sysconf*(`_SC_V6_ILP32_OFF32`) returns `-1`, the meaning of this value is unspecified.
15013 Otherwise, this value is the set of options to be given to the *lint* utility to check application
15014 source using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

15015 `_CS_V6_ILP32_OFFBIG_CFLAGS`

15016 If *sysconf*(`_SC_V6_ILP32_OFFBIG`) returns `-1`, the meaning of this value is unspecified.
15017 Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to
15018 build an application using a programming model with 32-bit **int**, **long**, and **pointer** types,

15019 and an **off_t** type using at least 64 bits.

15020 **_CS_V6_ILP32_OFFBIG_LDFLAGS**
15021 If *sysconf*(*_SC_V6_ILP32_OFFBIG*) returns -1, the meaning of this value is unspecified.
15022 Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build
15023 an application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
15024 **off_t** type using at least 64 bits.

15025 **_CS_V6_ILP32_OFFBIG_LIBS**
15026 If *sysconf*(*_SC_V6_ILP32_OFFBIG*) returns -1, the meaning of this value is unspecified.
15027 Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an
15028 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
15029 **off_t** type using at least 64 bits.

15030 **_CS_V6_ILP32_OFFBIG_LINTFLAGS**
15031 If *sysconf*(*_SC_V6_ILP32_OFFBIG*) returns -1, the meaning of this value is unspecified.
15032 Otherwise, this value is the set of options to be given to the *lint* utility to check an
15033 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
15034 **off_t** type using at least 64 bits.

15035 **_CS_V6_LP64_OFF64_CFLAGS**
15036 If *sysconf*(*_SC_V6_LP64_OFF64*) returns -1, the meaning of this value is unspecified.
15037 Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to
15038 build an application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t**
15039 types.

15040 **_CS_V6_LP64_OFF64_LDFLAGS**
15041 If *sysconf*(*_SC_V6_LP64_OFF64*) returns -1, the meaning of this value is unspecified.
15042 Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build
15043 an application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

15044 **_CS_V6_LP64_OFF64_LIBS**
15045 If *sysconf*(*_SC_V6_LP64_OFF64*) returns -1, the meaning of this value is unspecified.
15046 Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an
15047 application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

15048 **_CS_V6_LP64_OFF64_LINTFLAGS**
15049 If *sysconf*(*_SC_V6_LP64_OFF64*) returns -1, the meaning of this value is unspecified.
15050 Otherwise, this value is the set of options to be given to the *lint* utility to check application
15051 source using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

15052 **_CS_V6_LPBIG_OFFBIG_CFLAGS**
15053 If *sysconf*(*_SC_V6_LPBIG_OFFBIG*) returns -1, the meaning of this value is unspecified.
15054 Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to
15055 build an application using a programming model with an **int** type using at least 32 bits and
15056 **long**, **pointer**, and **off_t** types using at least 64 bits.

15057 **_CS_V6_LPBIG_OFFBIG_LDFLAGS**
15058 If *sysconf*(*_SC_V6_LPBIG_OFFBIG*) returns -1, the meaning of this value is unspecified.
15059 Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build
15060 an application using a programming model with an **int** type using at least 32 bits and **long**,
15061 **pointer**, and **off_t** types using at least 64 bits.

15062 **_CS_V6_LPBIG_OFFBIG_LIBS**
15063 If *sysconf*(*_SC_V6_LPBIG_OFFBIG*) returns -1, the meaning of this value is unspecified.
15064 Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an
15065 application using a programming model with an **int** type using at least 32 bits and **long**,

15066 **pointer**, and **off_t** types using at least 64 bits.

15067 `_CS_V6_LPBIG_OFFBIG_LINTFLAGS`
15068 If `sysconf(_SC_V6_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
15069 Otherwise, this value is the set of options to be given to the `lint` utility to check application
15070 source using a programming model with an **int** type using at least 32 bits and **long**, **pointer**,
15071 and **off_t** types using at least 64 bits.

15072 XSI `_CS_XBS5_ILP32_OFF32_CFLAGS (LEGACY)`
15073 If `sysconf(_SC_XBS5_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.
15074 Otherwise, this value is the set of initial options to be given to the `cc` and `c99` utilities to
15075 build an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t**
15076 types.

15077 XSI `_CS_XBS5_ILP32_OFF32_LDFLAGS (LEGACY)`
15078 If `sysconf(_SC_XBS5_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.
15079 Otherwise, this value is the set of final options to be given to the `cc` and `c99` utilities to build
15080 an application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

15081 XSI `_CS_XBS5_ILP32_OFF32_LIBS (LEGACY)`
15082 If `sysconf(_SC_XBS5_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.
15083 Otherwise, this value is the set of libraries to be given to the `cc` and `c99` utilities to build an
15084 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

15085 XSI `_CS_XBS5_ILP32_OFF32_LINTFLAGS (LEGACY)`
15086 If `sysconf(_SC_XBS5_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.
15087 Otherwise, this value is the set of options to be given to the `lint` utility to check application
15088 source using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

15089 XSI `_CS_XBS5_ILP32_OFFBIG_CFLAGS (LEGACY)`
15090 If `sysconf(_SC_XBS5_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
15091 Otherwise, this value is the set of initial options to be given to the `cc` and `c99` utilities to
15092 build an application using a programming model with 32-bit **int**, **long**, and **pointer** types,
15093 and an **off_t** type using at least 64 bits.

15094 XSI `_CS_XBS5_ILP32_OFFBIG_LDFLAGS (LEGACY)`
15095 If `sysconf(_SC_XBS5_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
15096 Otherwise, this value is the set of final options to be given to the `cc` and `c99` utilities to build
15097 an application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
15098 **off_t** type using at least 64 bits.

15099 XSI `_CS_XBS5_ILP32_OFFBIG_LIBS (LEGACY)`
15100 If `sysconf(_SC_XBS5_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
15101 Otherwise, this value is the set of libraries to be given to the `cc` and `c99` utilities to build an
15102 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
15103 **off_t** type using at least 64 bits.

15104 XSI `_CS_XBS5_ILP32_OFFBIG_LINTFLAGS (LEGACY)`
15105 If `sysconf(_SC_XBS5_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
15106 Otherwise, this value is the set of options to be given to the `lint` utility to check an
15107 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
15108 **off_t** type using at least 64 bits.

15109 XSI `_CS_XBS5_LP64_OFF64_CFLAGS (LEGACY)`
15110 If `sysconf(_SC_XBS5_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.
15111 Otherwise, this value is the set of initial options to be given to the `cc` and `c99` utilities to
15112 build an application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t**

15113 types.

15114 XSI **_CS_XBS5_LP64_OFF64_LDFLAGS (LEGACY)**
 15115 If *sysconf*(_SC_XBS5_LP64_OFF64) returns -1, the meaning of this value is unspecified.
 15116 Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build
 15117 an application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

15118 XSI **_CS_XBS5_LP64_OFF64_LIBS (LEGACY)**
 15119 If *sysconf*(_SC_XBS5_LP64_OFF64) returns -1, the meaning of this value is unspecified.
 15120 Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an
 15121 application using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

15122 XSI **_CS_XBS5_LP64_OFF64_LINTFLAGS (LEGACY)**
 15123 If *sysconf*(_SC_XBS5_LP64_OFF64) returns -1, the meaning of this value is unspecified.
 15124 Otherwise, this value is the set of options to be given to the *lint* utility to check application
 15125 source using a programming model with 64-bit **int**, **long**, **pointer**, and **off_t** types.

15126 XSI **_CS_XBS5_LPBIG_OFFBIG_CFLAGS (LEGACY)**
 15127 If *sysconf*(_SC_XBS5_LPBIG_OFFBIG) returns -1, the meaning of this value is unspecified.
 15128 Otherwise, this value is the set of initial options to be given to the *cc* and *c99* utilities to
 15129 build an application using a programming model with an **int** type using at least 32 bits and
 15130 **long**, **pointer**, and **off_t** types using at least 64 bits.

15131 XSI **_CS_XBS5_LPBIG_OFFBIG_LDFLAGS (LEGACY)**
 15132 If *sysconf*(_SC_XBS5_LPBIG_OFFBIG) returns -1, the meaning of this value is unspecified.
 15133 Otherwise, this value is the set of final options to be given to the *cc* and *c99* utilities to build
 15134 an application using a programming model with an **int** type using at least 32 bits and **long**,
 15135 **pointer**, and **off_t** types using at least 64 bits.

15136 XSI **_CS_XBS5_LPBIG_OFFBIG_LIBS (LEGACY)**
 15137 If *sysconf*(_SC_XBS5_LPBIG_OFFBIG) returns -1, the meaning of this value is unspecified.
 15138 Otherwise, this value is the set of libraries to be given to the *cc* and *c99* utilities to build an
 15139 application using a programming model with an **int** type using at least 32 bits and **long**,
 15140 **pointer**, and **off_t** types using at least 64 bits.

15141 XSI **_CS_XBS5_LPBIG_OFFBIG_LINTFLAGS (LEGACY)**
 15142 If *sysconf*(_SC_XBS5_LPBIG_OFFBIG) returns -1, the meaning of this value is unspecified.
 15143 Otherwise, this value is the set of options to be given to the *lint* utility to check application
 15144 source using a programming model with an **int** type using at least 32 bits and **long**, **pointer**,
 15145 and **off_t** types using at least 64 bits.

15146 The following symbolic constants shall be defined for the *lseek*() and *fcntl*() functions (they have
 15147 distinct values):

15148 {SEEK_CUR} Set file offset to current plus *offset*.
 15149 {SEEK_END} Set file offset to EOF plus *offset*.
 15150 {SEEK_SET} Set file offset to *offset*.

15151 The following symbolic constants shall be defined for *sysconf*():

15152 **_SC_2_C_BIND**
 15153 **_SC_2_C_DEV**
 15154 **_SC_2_C_VERSION**
 15155 **_SC_2_FORT_DEV**
 15156 **_SC_2_FORT_RUN**
 15157 **_SC_2_LOCALEDEF**
 15158 **_SC_2_PBS**


```

15159     _SC_2_PBS_ACCOUNTING
15160     _SC_2_PBS_CHECKPOINT
15161     _SC_2_PBS_LOCATE
15162     _SC_2_PBS_MESSAGE
15163     _SC_2_PBS_TRACK
15164     _SC_2_SW_DEV
15165     _SC_2_UPE
15166     _SC_2_VERSION
15167     _SC_ARG_MAX
15168     _SC_AIO_LISTIO_MAX
15169     _SC_AIO_MAX
15170     _SC_AIO_PRIO_DELTA_MAX
15171     _SC_ASYNCHRONOUS_IO
15172 XSI   _SC_ATEXIT_MAX
15173 BAR   _SC_BARRIERS
15174     _SC_BASE
15175     _SC_BC_BASE_MAX
15176     _SC_BC_DIM_MAX
15177     _SC_BC_SCALE_MAX
15178     _SC_BC_STRING_MAX
15179     _SC_C_LANG_SUPPORT
15180     _SC_C_LANG_SUPPORT_R
15181     _SC_CHILD_MAX
15182     _SC_CLK_TCK
15183 CS    _SC_CLOCK_SELECTION
15184     _SC_COLL_WEIGHTS_MAX
15185     _SC_DELAYTIMER_MAX
15186     _SC_DEVICE_IO
15187     _SC_DEVICE_SPECIFIC
15188     _SC_DEVICE_SPECIFIC_R
15189     _SC_EXPR_NEST_MAX
15190     _SC_FD_MGMT
15191     _SC_FIFO
15192     _SC_FILE_ATTRIBUTES
15193     _SC_FILE_LOCKING
15194     _SC_FILE_SYSTEM
15195     _SC_FSYNC
15196     _SC_GETGR_R_SIZE_MAX
15197     _SC_GETPW_R_SIZE_MAX
15198 XSI   _SC_IOV_MAX
15199     _SC_JOB_CONTROL
15200     _SC_LINE_MAX
15201     _SC_LOGIN_NAME_MAX
15202     _SC_MAPPED_FILES
15203     _SC_MEMLOCK
15204     _SC_MEMLOCK_RANGE
15205     _SC_MEMORY_PROTECTION
15206     _SC_MESSAGE_PASSING
15207 MON   _SC_MONOTONIC_CLOCK
15208     _SC_MQ_OPEN_MAX
15209     _SC_MQ_PRIO_MAX
15210     _SC_MULTIPLE_PROCESS

```

15211		_SC_NETWORKING
15212		_SC_NGROUPS_MAX
15213		_SC_OPEN_MAX
15214	XSI	_SC_PAGE_SIZE
15215		_SC_PAGESIZE
15216		_SC_PIPE
15217		_SC_PRIORITIZED_IO
15218		_SC_PRIORITY_SCHEDULING
15219		_SC_RE_DUP_MAX
15220	THR	_SC_READER_WRITER_LOCKS
15221		_SC_REALTIME_SIGNALS
15222		_SC_REGEX
15223		_SC_RTSIG_MAX
15224		_SC_SAVED_IDS
15225		_SC_SEMAPHORES
15226		_SC_SEM_NSEMS_MAX
15227		_SC_SEM_VALUE_MAX
15228		_SC_SHARED_MEMORY_OBJECTS
15229		_SC_SHELL
15230		_SC_SIGNALS
15231		_SC_SIGQUEUE_MAX
15232		_SC_SINGLE_PROCESS
15233	SPI	_SC_SPIN_LOCKS
15234		_SC_STREAM_MAX
15235		_SC_SYNCHRONIZED_IO
15236		_SC_SYSTEM_DATABASE
15237		_SC_SYSTEM_DATABASE_R
15238		_SC_THREAD_ATTR_STACKADDR
15239		_SC_THREAD_ATTR_STACKSIZE
15240		_SC_THREAD_DESTRUCTOR_ITERATIONS
15241		_SC_THREAD_KEYS_MAX
15242		_SC_THREAD_PRIO_INHERIT
15243		_SC_THREAD_PRIO_PROTECT
15244		_SC_THREAD_PRIORITY_SCHEDULING
15245		_SC_THREAD_PROCESS_SHARED
15246		_SC_THREAD_SAFE_FUNCTIONS
15247		_SC_THREAD_STACK_MIN
15248		_SC_THREAD_THREADS_MAX
15249		_SC_THREADS
15250		_SC_TIMER_MAX
15251		_SC_TIMERS
15252	TRC	_SC_TRACE
15253	TEF	_SC_TRACE_EVENT_FILTER
15254	TRL	_SC_TRACE_LOG
15255	TRI	_SC_TRACE_INHERIT
15256		_SC_TTY_NAME_MAX
15257	TYM	_SC_TYPED_MEMORY_OBJECTS
15258		_SC_TZNAME_MAX
15259		_SC_USER_GROUPS
15260		_SC_USER_GROUPS_R
15261		_SC_V6_ILP32_OFF32
15262		_SC_V6_ILP32_OFFBIG

```

15263     _SC_V6_LP64_OFF64
15264     _SC_V6_LP64_OFF64
15265     _SC_VERSION
15266 XSI   _SC_XBS5_ILP32_OFF32 (LEGACY)
15267     _SC_XBS5_ILP32_OFFBIG (LEGACY)
15268     _SC_XBS5_LP64_OFF64 (LEGACY)
15269     _SC_XBS5_LP64_OFF64 (LEGACY)
15270     _SC_XOPEN_CRYPT
15271     _SC_XOPEN_ENH_I18N
15272     _SC_XOPEN_LEGACY
15273     _SC_XOPEN_REALTIME
15274     _SC_XOPEN_REALTIME_THREADS
15275     _SC_XOPEN_SHM
15276     _SC_XOPEN_STREAMS
15277     _SC_XOPEN_UNIX
15278     _SC_XOPEN_VERSION
15279     _SC_XOPEN_XCU_VERSION
15280

```

15281 The two constants `_SC_PAGESIZE` and `_SC_PAGE_SIZE` may be defined to have the same
15282 value.

15283 The following symbolic constants shall be defined as possible values for the *function* argument
15284 to the `lockf()` function:

```

15285     F_LOCK           Lock a section for exclusive use.
15286     F_TEST           Test section for locks by other processes.
15287     F_TLOCK          Test and lock a section for exclusive use.
15288     F_ULOCK          Unlock locked sections.

```

15289 The following symbolic constants shall be defined for `pathconf()`:

```

15290 ADV   _PC_ALLOC_SIZE_MIN
15291 AIO   _PC_ASYNC_IO
15292     _PC_CHOWN_RESTRICTED
15293     _PC_FILESIZEBITS
15294     _PC_LINK_MAX
15295     _PC_MAX_CANON
15296     _PC_MAX_INPUT
15297     _PC_NAME_MAX
15298     _PC_NO_TRUNC
15299     _PC_PATH_MAX
15300     _PC_PIPE_BUF
15301     _PC_PRIO_IO
15302 ADV   _PC_REC_INCR_XFER_SIZE
15303     _PC_REC_MAX_XFER_SIZE
15304     _PC_REC_MIN_XFER_SIZE
15305     _PC_REC_XFER_ALIGN
15306     _PC_SYNC_IO
15307     _PC_VDISABLE

```

15308 The following symbolic constants shall be defined for file streams:

15309 `STDERR_FILENO` File number of *stderr*; 2.

15310 `STDIN_FILENO` File number of *stdin*; 0.

15311 `STDOUT_FILENO` File number of *stdout*; 1.

15312 **Type Definitions**

15313 The `size_t`, `ssize_t`, `uid_t`, `gid_t`, `off_t`, and `pid_t` types shall be defined as described in
15314 <sys/types.h>.

15315 The `useconds_t` type shall be defined as described in <sys/types.h>.

15316 The `intptr_t` type shall be defined as described in <inttypes.h>.

15317 The `socklen_t` type shall be defined as described in <sys/socket.h>.

15318 **Declarations**

15319 The following shall be declared as functions and may also be defined as macros. Function
15320 prototypes shall be provided for use with an ISO C standard compiler.

```

15321 int      access(const char *, int);
15322 unsigned alarm(unsigned);
15323 XSI int   brk(void *);
15324 int     chdir(const char *);
15325 int     chown(const char *, uid_t, gid_t);
15326 int     close(int);
15327 size_t  confstr(int, char *, size_t);
15328 XSI char *crypt(const char *, const char *);
15329 char   *ctermid(char *);
15330 int     dup(int);
15331 int     dup2(int, int);
15332 XSI void encrypt(char[64], int);
15333 int     execl(const char *, const char *, ...);
15334 int     execlp(const char *, const char *, ...);
15335 int     execlp(const char *, const char *, ...);
15336 int     execv(const char *, char *const []);
15337 int     execve(const char *, char *const [], char *const []);
15338 int     execvp(const char *, char *const []);
15339 void    _exit(int);
15340 XSI int  fchown(int, uid_t, gid_t);
15341 int     fchdir(int);
15342 SIO int  fdatsync(int);
15343 pid_t   fork(void);
15344 long    fpathconf(int, int);
15345 int     fsync(int);
15346 int     ftruncate(int, off_t);
15347 char   *getcwd(char *, size_t);
15348 gid_t   getegid(void);
15349 uid_t   geteuid(void);
15350 gid_t   getgid(void);
15351 int     getgroups(int, gid_t []);
15352 XSI long gethostid(void);
15353 int     gethostname(char *, socklen_t);

```

```

15354     char          *getlogin(void);
15355     int           getlogin_r(char *, size_t);
15356     int           getopt(int, char * const [], const char *);
15357 XSI     pid_t      getpgid(pid_t);
15358     pid_t      getpgrp(void);
15359     pid_t      getpid(void);
15360     pid_t      getppid(void);
15361 XSI     pid_t      getsid(pid_t);
15362     uid_t      getuid(void);
15363 XSI     char          *getwd(char *); (LEGACY)
15364     int           isatty(int);
15365 XSI     int           lchown(const char *, uid_t, gid_t);
15366     int           link(const char *, const char *);
15367 XSI     int           lockf(int, int, off_t);
15368     off_t      lseek(int, off_t, int);
15369 XSI     int           nice(int);
15370     long        pathconf(const char *, int);
15371     int           pause(void);
15372     int           pipe(int [2]);
15373 XSI     ssize_t     pread(int, void *, size_t, off_t);
15374     ssize_t     pwrite(int, const void *, size_t, off_t);
15375     ssize_t     read(int, void *, size_t);
15376     ssize_t     readlink(const char *restrict, char *restrict, size_t);
15377     int           rmdir(const char *);
15378 XSI     void          *sbrk(intptr_t);
15379     int           setegid(gid_t);
15380     int           seteuid(uid_t);
15381     int           setgid(gid_t);
15382     int           setpgid(pid_t, pid_t);
15383 XSI     pid_t      setpgrp(void);
15384     int           setregid(gid_t, gid_t);
15385     int           setreuid(uid_t, uid_t);
15386     pid_t      setsid(void);
15387     int           setuid(uid_t);
15388     unsigned    sleep(unsigned);
15389 XSI     void          swab(const void *restrict, void *restrict, ssize_t);
15390     int           symlink(const char *, const char *);
15391     void          sync(void);
15392     long        sysconf(int);
15393     pid_t      tcgetpgrp(int);
15394     int           tcsetpgrp(int, pid_t);
15395 XSI     int           truncate(const char *, off_t);
15396     char          *ttyname(int);
15397     int           ttyname_r(int, char *, size_t);
15398 XSI     useconds_t  ualarm(useconds_t, useconds_t);
15399     int           unlink(const char *);
15400 XSI     int           usleep(useconds_t);
15401     pid_t      vfork(void);
15402     ssize_t     write(int, const void *, size_t);

```

15403 Implementations may also include the *pthread_atfork()* prototype as defined in <pthread.h> (on
15404 page 322).

15405 The following external variables shall be declared:

```
15406 extern char *optarg;
15407 extern int optind, opterr, optopt;
```

15408 APPLICATION USAGE

15409 None.

15410 RATIONALE

15411 As IEEE Std. 1003.1-200x evolved, certain options became sufficiently standardized that it was
15412 concluded that simply requiring one of the option choices was simpler than retaining the option.
15413 However, for backwards compatibility, the option flags (with required constant values) are
15414 retained.

15415 Version Test Macros

15416 The standard developers considered altering the definition of `_POSIX_VERSION` and removing
15417 `_SC_VERSION` from the specification of `sysconf()` since the utility to an application was deemed
15418 by some to be minimal, and since the implementation of the functionality is potentially
15419 problematic.

15420 Applications are allowed the ability to adapt to various versions of IEEE Std. 1003.1-200x at
15421 compile time by conditionally compiling different code depending on the value of
15422 `_POSIX_VERSION`. For example, an application which expects the semantics of the
15423 POSIX.1-1988 standard `cuserid()` but also wishes to compile and run on a system which
15424 conforms to the POSIX.1-1990 standard might be coded as in the following program fragment:

```
15425 #if _POSIX_VERSION == 198808L
15426     val = cuserid();
15427 #else
15428     {
15429     struct passwd *pwp;
15430     pwp = getpwuid(geteuid());
15431     val = pwp->pw_name;
15432     }
15433 #endif
```

15434 While POSIX does not make any attempt to define application binary interaction with the
15435 underlying operating system, the standard developers recognized that support for existing
15436 application binaries is a concern to manufacturers, application developers, and the users of
15437 implementations conforming to IEEE Std. 1003.1-200x. To that end, an application can query
15438 `_SC_VERSION` at runtime via `sysconf()` to determine whether the current version of the
15439 operating system supports the necessary functionality as in the following program fragment:

```
15440 if(sysconf(_SC_VERSION) != 200xxxL) {
15441     fprintf(stderr, "POSIX.1-1990 system required, terminating\n")
15442     exit(1);
15443 }
```

15444 While the above example does not provide the greatest degree of imaginable utility to the
15445 application developer or user, it is arguably better than a core dump or some other equally
15446 obscure result. (It is also possible for implementations to encode and recognize application
15447 binaries compiled in various POSIX.1-conforming environments, and modify the semantics of
15448 the underlying system to conform to the expectations of the application.) For the reasons
15449 outlined in the preceding paragraphs, the standard developers elected to retain the
15450 `_POSIX_VERSION` and `_SC_VERSION` functionality.

15451 **Compile-Time Symbolic Constants for System-Wide Options**

15452 IEEE Std. 1003.1-200x now includes support in certain areas for the newly adopted policy
15453 governing options and stubs.

15454 This policy provides flexibility for implementations in how they support options. It also
15455 specifies how conforming applications can adapt to different implementations that support
15456 different sets of options. It allows the following:

- 15457 1. If an implementation has no interest in supporting an option, it does not have to provide
15458 anything associated with that option beyond the announcement that it does not support it.
- 15459 2. An implementation can support a partial or incompatible version of an option (as a non-
15460 standard extension) as long as it does not claim to support the option.
- 15461 3. An application can determine whether the option is supported. A strictly conforming
15462 application must check this announcement mechanism before first using anything
15463 associated with the option.

15464 There is an important implication of this policy. IEEE Std. 1003.1-200x cannot dictate the
15465 behavior of interfaces associated with an option when the implementation does not claim to
15466 support the option. In particular, it cannot require that a function associated with an
15467 unsupported option will fail if it does not perform as specified. However, this policy does not
15468 prevent a standard from requiring certain functions to always be present, but that they shall
15469 always fail on some implementations. The *setpgid()* function in the POSIX.1-1990 standard, for
15470 example, is considered appropriate.

15471 The POSIX standards include various options, and the C language binding support for an option
15472 implies that the implementation must supply data types and function interfaces. An application
15473 must be able to discover whether the implementation supports each option.

15474 Any application must consider the following three cases for each option:

- 15475 1. Option never supported.

15476 The implementation advertises at compile time that the option will never be supported. In
15477 this case, it is not necessary for the implementation to supply any of the data types or
15478 function interfaces that are provided only as part of the option. The implementation might
15479 provide data types and functions that are similar to those defined by IEEE Std. 1003.1-200x,
15480 but there is no guarantee for any particular behavior.

- 15481 2. Option always supported.

15482 The implementation advertises at compile time that the option will always be supported.
15483 In this case, all data types and function interfaces shall be available and shall operate as
15484 specified.

- 15485 3. Option might or might not be supported.

15486 Some implementations might not provide a mechanism to specify support of options at
15487 compile time. In addition, the implementation might be unable or unwilling to specify
15488 support or non-support at compile time. In either case, any application that might use the
15489 option at runtime must be able to compile and execute. The implementation must provide,
15490 at compile time, all data types and function interfaces that are necessary to allow this. In
15491 this situation, there must be a mechanism that allows the application to query, at runtime,
15492 whether the option is supported. If the application attempts to use the option when it is
15493 not supported, the result is unspecified unless explicitly specified otherwise in
15494 IEEE Std. 1003.1-200x.

15495 **FUTURE DIRECTIONS**

15496 None.

15497 **SEE ALSO**

15498 <inttypes.h>, <limits.h>, <sys/socket.h>, <sys/types.h>, <termios.h>, the System Interfaces
 15499 volume of IEEE Std. 1003.1-200x, *access()*, *alarm()*, *chdir()*, *chown()*, *close()*, *crypt()*, *ctermid()*,
 15500 *dup()*, *encrypt()*, *environ()*, *exec()*, *exit()*, *fchdir()*, *fchown()*, *fcntl()*, *fork()*, *fpathconf()*, *fsync()*,
 15501 *ftruncate()*, *getcwd()*, *getegid()*, *geteuid()*, *getgid()*, *getgroups()*, *gethostid()*, *gethostname()*,
 15502 *getlogin()*, *getpgid()*, *getpgrp()*, *getpid()*, *getppid()*, *getsid()*, *getuid()*, *isatty()*, *lchown()*, *link()*,
 15503 *lockf()*, *lseek()*, *nice()*, *pathconf()*, *pause()*, *pipe()*, *read()*, *readlink()*, *rmdir()*, *setgid()*, *setpgid()*,
 15504 *setpgrp()*, *setregid()*, *setreuid()*, *setsid()*, *setuid()*, *sleep()*, *swab()*, *symlink()*, *sync()*, *sysconf()*,
 15505 *tcgetpgrp()*, *tcsetpgrp()*, *truncate()*, *ttyname()*, *ualarm()*, *unlink()*, *usleep()*, *vfork()*, *write()*

15506 **CHANGE HISTORY**

15507 First released in Issue 1. Derived from Issue 1 of the SVID.

15508 **Issue 4**15509 The symbolic constants `F_ULOCK`, `F_LOCK`, `F_TLOCK`, `F_TEST`, `GF_PATH`, `IF_PATH`, and
 15510 `PF_PATH` are withdrawn.15511 The required value of `_XOPEN_VERSION` is defined and the constant is marked as an extension.15512 The constants `_XOPEN_XPG2`, `_XOPEN_XPG3`, and `_XOPEN_XPG4` are added.15513 The constants `_POSIX2_*` are added.15514 Reference to the <sys/types.h> header is added for the definitions of `size_t`, `ssize_t`, `uid_t`, `gid_t`,
 15515 `off_t`, and `pid_t`. These are marked as extensions.15516 The names *chroot()*, *crypt()*, *encrypt()*, *fsync()*, *getopt()*, *getpass()*, *nice()*, and *swab()* are added to
 15517 the list of functions declared in this header. With the exception of *getopt()*, these are all marked
 15518 as extensions.

15519 The APPLICATION USAGE section is removed.

15520 The following changes are incorporated for alignment with the ISO POSIX-1 standard and the
 15521 ISO POSIX-2 standard:

- 15522 • The function declarations in this header are expanded to full ISO C standard prototypes.
- 15523 • A large number of new constants are defined for the *sysconf()* function, including all those
- 15524 with prefixes `_SC_2` and `_SC_BC`, plus:

15525 `_SC_COLL_WEIGHTS_MAX`15526 `_SC_EXPR_NEST_MAX`15527 `_SC_LINE_MAX`15528 `_SC_RE_DUP_MAX`15529 `_SC_STREAM_MAX`15530 `_SC_TZNAME_MAX`

- 15531 • The *confstr()* function is added to the list of functions declared in this header, complete with
- 15532 a new set of constants for alignment with the ISO POSIX-2 standard.

15533 The following change is incorporated for alignment with the FIPS requirements:

- 15534 • The following symbolic constants are always defined:

15535 _POSIX_CHOWN_RESTRICTED
 15536 _POSIX_NO_TRUNC
 15537 _POSIX_VDISABLE
 15538 _POSIX_SAVED_IDS
 15539 _POSIX_JOB_CONTROL

15540 In Issue 3, they are only defined if the associated option is present.

15541 **Issue 4, Version 2**

15542 The following changes are incorporated for X/OPEN UNIX conformance:

- 15543 • The Option Group constant `_XOPEN_UNIX` is defined.
- 15544 • The `sysconf()` symbolic constants `_SC_ATEXIT_MAX`, `_SC_IOV_MAX`, `_SC_PAGESIZE`, and
 15545 `_SC_PAGE_SIZE` are defined.
- 15546 • The `brk()`, `fchown()`, `fchdir()`, `ftruncate()`, `gethostid()`, `getpagesize()`, `getpgid()`, `getsid()`, `getwd()`,
 15547 `lchown()`, `lockf()`, `readlink()`, `sbrk()`, `setpgrp()`, `setregid()`, `setreuid()`, `symlink()`, `sync()`,
 15548 `truncate()`, `ualarm()`, `usleep()`, and `vfork()` functions are added to the list of functions
 15549 declared in this header.
- 15550 • The symbolic constants `F_ULOCK`, `F_LOCK`, `F_TLOCK`, and `F_TEST` are added.

15551 **Issue 5**

15552 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 15553 Threads Extension.

15554 The symbolic constants `_XOPEN_REALTIME` and `_XOPEN_REALTIME_THREADS` are added.
 15555 `_POSIX2_C_BIND`, `_XOPEN_ENH_I18N`, and `_XOPEN_SHM` must now be set to a value other
 15556 than `-1` by a conforming implementation.

15557 Large File System extensions are added.

15558 The type of the argument to `sbrk()` is changed from `int` to `intptr_t`.

15559 `_XBS_` constants are added to the list of constants for Options and Option Groups, to the list of
 15560 constants for the `confstr()` function, and to the list of constants to the `sysconf()` function. These
 15561 are all marked EX.

15562 **Issue 6**

15563 `_POSIX2_C_VERSION` is removed.

15564 The Open Group corrigenda item U026/4 has been applied, adding the prototype for `fdatasync()`.

15565 The Open Group corrigenda item U026/1 has been applied, adding the symbols
 15566 `_SC_XOPEN_LEGACY`, `_SC_XOPEN_REALTIME`, and `_SC_XOPEN_REALTIME_THREADS`.

15567 The symbols `_XOPEN_STREAMS` and `_SC_XOPEN_STREAMS` are added to support the XSI
 15568 STREAMS Option Group.

15569 Constants for profiling options are added.

15570 Text in the DESCRIPTION relating to conformance requirements is moved elsewhere in
 15571 IEEE Std. 1003.1-200x.

15572 The legacy symbol `_SC_PASS_MAX` is removed.

15573 The following new requirements on POSIX implementations derive from alignment with the
 15574 Single UNIX Specification:

- 15575 • The `_CS_XBS5_*` constants are added for the `confstr()` function.

- 15576 • The `_SC_XBS5_*` constants are added for the `sysconf()` function.
- 15577 • The symbolic constants `F_ULOCK`, `F_LOCK`, `F_TLOCK`, and `F_TEST` are added.
- 15578 • The `uid_t`, `gid_t`, `off_t`, `pid_t`, and `useconds_t` types are mandated.
- 15579 The `gethostname()` prototype is added for sockets.
- 15580 New section added for System Wide Options.
- 15581 Function prototypes for `setegid()` and `seteuid()` are added.
- 15582 Option symbolic constants are added for `_POSIX_ADVISORY_INFO`, `_POSIX_CPUTIME`,
15583 `_POSIX_SPAWN`, `_POSIX_SPORADIC_SERVER`, `_POSIX_THREAD_CPUTIME`,
15584 `_POSIX_THREAD_SPORADIC_SERVER`, and `_POSIX_TIMEOUTS`, and `pathconf()` variables are
15585 added for `_PC_ALLOC_SIZE_MIN`, `_PC_REC_INCR_XFER_SIZE`, `_PC_REC_MAX_XFER_SIZE`,
15586 `_PC_REC_MIN_XFER_SIZE`, and `_PC_REC_XFER_ALIGN` for alignment with
15587 IEEE Std. 1003.1d-1999.
- 15588 The following are added for alignment with IEEE Std. 1003.1j-2000:
 - 15589 • Option symbolic constants `_POSIX_BARRIERS`, `_POSIX_CLOCK_SELECTION`,
15590 `_POSIX_MONOTONIC_CLOCK`, `_POSIX_READER_WRITER_LOCKS`,
15591 `_POSIX_SPIN_LOCKS`, and `_POSIX_TYPED_MEMORY_OBJECTS`
 - 15592 • `sysconf()` variables `_SC_BARRIERS`, `_SC_CLOCK_SELECTION`,
15593 `_SC_MONOTONIC_CLOCK`, `_SC_READER_WRITER_LOCKS`, `_SC_SPIN_LOCKS`, and
15594 `_SC_TYPED_MEMORY_OBJECTS`
- 15595 The `_SC_XBS5` macros associated with the ISO/IEC 9899:1990 standard are marked `LEGACY`,
15596 and new equivalent `_SC_V6` macros associated with the ISO/IEC 9899:1999 standard are
15597 introduced.
- 15598 The `getwd()` function is marked `LEGACY`.
- 15599 The `restrict` keyword is added to the prototypes for `realink()` and `swab()`.
- 15600 Constants for options are now harmonized, so when supported they take the year of approval of
15601 IEEE Std. 1003.1-200x as the value.
- 15602 The following are added for alignment with IEEE Std. 1003.1q-2000:
 - 15603 • Optional symbolic constants `_POSIX_TRACE`, `_POSIX_TRACE_EVENT_FILTER`,
15604 `_POSIX_TRACE_LOG`, and `_POSIX_TRACE_INHERIT`
 - 15605 • The `sysconf()` symbolic constants `_SC_TRACE`, `_SC_TRACE_EVENT_FILTER`,
15606 `_SC_TRACE_LOG`, and `_SC_TRACE_INHERIT`.

15607 **NAME**

15608 utime.h — access and modification times structure

15609 **SYNOPSIS**

15610 #include <utime.h>

15611 **DESCRIPTION**

15612 The <utime.h> header shall declare the structure **utimbuf**, which shall include the following
15613 members:

15614 time_t actime Access time.

15615 time_t modtime Modification time.

15616 The times shall be measured in seconds since the Epoch.

15617 The type **time_t** shall be defined as described in <sys/types.h>.

15618 The following shall be declared as a function and may also be defined as a macro. Function
15619 prototypes shall be provided for use with an ISO C standard compiler.

15620 int utime(const char *, const struct utimbuf *);

15621 **APPLICATION USAGE**

15622 None.

15623 **RATIONALE**

15624 None.

15625 **FUTURE DIRECTIONS**

15626 None.

15627 **SEE ALSO**

15628 <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *utime()*

15629 **CHANGE HISTORY**

15630 First released in Issue 3.

15631 **Issue 4**

15632 Reference to the <sys/types.h> header is added for the definition of **time_t**. This is marked as an
15633 extension.

15634 The following change is incorporated for alignment with the ISO POSIX-1 standard:

- 15635 • The function declarations in this header are expanded to full ISO C standard prototypes.

15636 **Issue 6**

15637 The following new requirements on POSIX implementations derive from alignment with the
15638 Single UNIX Specification:

- 15639 • The **time_t** type is defined.

15640 **NAME**

15641 utmpx.h — user accounting database definitions

15642 **SYNOPSIS**15643 XSI `#include <utmpx.h>`

15644

15645 **DESCRIPTION**15646 The **<utmpx.h>** header shall define the **utmpx** structure that shall include at least the following
15647 members:

15648	char	ut_user[]	User login name.
15649	char	ut_id[]	Unspecified initialization process identifier.
15650	char	ut_line[]	Device name.
15651	pid_t	ut_pid	Process ID.
15652	short	ut_type	Type of entry.
15653	struct timeval	ut_tv	Time entry was made.

15654 The **pid_t** type shall be defined through **typedef** as described in **<sys/types.h>**.15655 The **timeval** structure shall be defined as described in **<sys/time.h>**.15656 Inclusion of the **<utmpx.h>** header may also make visible all symbols from **<sys/time.h>**.15657 The following symbolic constants shall be defined as possible values for the *ut_type* member of
15658 the **utmpx** structure:

15659	EMPTY	No valid user accounting information.
15660	BOOT_TIME	Identifies time of system boot.
15661	OLD_TIME	Identifies time when system clock changed.
15662	NEW_TIME	Identifies time after system clock changed.
15663	USER_PROCESS	Identifies a process.
15664	INIT_PROCESS	Identifies a process spawned by the init process.
15665	LOGIN_PROCESS	Identifies the session leader of a logged in user.
15666	DEAD_PROCESS	Identifies a session leader who has exited.

15667 The following shall be declared as functions and may also be defined as macros. Function
15668 prototypes shall be provided for use with an ISO C standard compiler.

15669	void	endutxent(void);
15670	struct utmpx	*getutxent(void);
15671	struct utmpx	*getutxid(const struct utmpx *);
15672	struct utmpx	*getutxline(const struct utmpx *);
15673	struct utmpx	*pututxline(const struct utmpx *);
15674	void	setutxent(void);

15675 **APPLICATION USAGE**

15676 None.

15677 **RATIONALE**

15678 None.

15679 **FUTURE DIRECTIONS**

15680 None.

15681 **SEE ALSO**15682 <sys/time.h>, <sys/types.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *endutxent()*15683 **CHANGE HISTORY**

15684 First released in Issue 4, Version 2.

15685 **NAME**15686 **wchar.h** — wide-character types15687 **SYNOPSIS**

15688 #include <wchar.h>

15689 **DESCRIPTION**

15690 **CX** The functionality described on this reference page extends the ISO C standard. Applications
 15691 shall define the appropriate feature test macro (see the System Interfaces volume of
 15692 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 15693 symbols in this header.

15694 The <**wchar.h**> header shall define the following data types through **typedef**:

15695 **wchar_t** As described in <**stddef.h**>.

15696 **wint_t** An integer type capable of storing any valid value of **wchar_t** or WEOF.

15697 **wctype_t** A scalar type of a data object that can hold values which represent locale-
 15698 specific character classification.

15699 **mbstate_t** An object type other than an array type that can hold the conversion state
 15700 information necessary to convert between sequences of (possibly multibyte)
 15701 **XSI** characters and wide characters. If a codeset is being used such that an
 15702 **mbstate_t** needs to preserve more than 2 levels of reserved state, the results
 15703 are unspecified.

15704 **XSI** **FILE** As described in <**stdio.h**>.

15705 **size_t** As described in <**stddef.h**>.

15706 The <**wchar.h**> header shall declare the following as functions and may also define them as
 15707 macros. Function prototypes shall be provided for use with an ISO C standard compiler.

```

15708        wint_t        btowc(int);
15709        wint_t        fgetwc(FILE *);
15710        wchar_t       *fgetws(wchar_t *restrict, int, FILE *restrict);
15711        wint_t        fputwc(wchar_t, FILE *);
15712        int         fputws(const wchar_t *restrict, FILE *restrict);
15713        int         fwide(FILE *, int);
15714        int         fwprintf(FILE *restrict, const wchar_t *restrict, ...);
15715        int         fwscanf(FILE *restrict, const wchar_t *restrict, ...);
15716        wint_t        getwc(FILE *);
15717        wint_t        getwchar(void);
15718        int         iswalnum(wint_t);
15719        int         iswalpha(wint_t);
15720        int         iswcntrl(wint_t);
15721        int         iswctype(wint_t, wctype_t);
15722        int         iswdigit(wint_t);
15723        int         iswgraph(wint_t);
15724        int         iswlower(wint_t);
15725        int         iswprint(wint_t);
15726        int         iswpunct(wint_t);
15727        int         iswspace(wint_t);
15728        int         iswupper(wint_t);
15729        int         iswxdigit(wint_t);
15730        size_t        mbrlen(const char *restrict, size_t, mbstate_t *restrict);
15731        size_t        mbrtowc(wchar_t *restrict, const char *restrict, size_t,
```

```

15732         mbstate_t *restrict);
15733 int      mbsinit(const mbstate_t *);
15734 size_t   mbsrtowcs(wchar_t *restrict, const char **restrict, size_t,
15735         mbstate_t *restrict);
15736 wint_t   putwc(wchar_t, FILE *);
15737 wint_t   putwchar(wchar_t);
15738 int      swprintf(wchar_t *restrict, size_t,
15739         const wchar_t *restrict, ...);
15740 int      swscanf(const wchar_t *restrict,
15741         const wchar_t *restrict, ...);
15742 wint_t   tolower(wint_t);
15743 wint_t   toupper(wint_t);
15744 wint_t   ungetwc(wint_t, FILE *);
15745 int      vfwprintf(FILE *restrict, const wchar_t *restrict, va_list);
15746 int      vwscanf(FILE *restrict, const wchar_t *restrict, va_list);
15747 int      vwprintf(const wchar_t *restrict, va_list);
15748 int      vswprintf(wchar_t *restrict, size_t,
15749         const wchar_t *restrict, va_list);
15750 int      vswscanf(const wchar_t *restrict, const wchar_t *restrict,
15751         va_list);
15752 int      vwscanf(const wchar_t *restrict, va_list);
15753 size_t   wcrntomb(char *restrict, wchar_t, mbstate_t *restrict);
15754 wchar_t  *wcsat(wchar_t *restrict, const wchar_t *restrict);
15755 wchar_t  *wcschr(const wchar_t *, wchar_t);
15756 int      wcscmp(const wchar_t *, const wchar_t *);
15757 int      wcscoll(const wchar_t *, const wchar_t *);
15758 wchar_t  *wcscpy(wchar_t *restrict, const wchar_t *restrict);
15759 size_t   wcsncpy(const wchar_t *, const wchar_t *);
15760 size_t   wcsftime(wchar_t *restrict, size_t,
15761         const wchar_t *restrict, const struct tm *restrict);
15762 size_t   wcslen(const wchar_t *);
15763 wchar_t  *wcsncat(wchar_t *restrict, const wchar_t *restrict, size_t);
15764 int      wcsncmp(const wchar_t *, const wchar_t *, size_t);
15765 wchar_t  *wcsncpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15766 wchar_t  *wcpbr(const wchar_t *, const wchar_t *);
15767 wchar_t  *wcsrchr(const wchar_t *, wchar_t);
15768 size_t   wcsrtombs(char *restrict, const wchar_t **restrict,
15769         size_t, mbstate_t *restrict);
15770 size_t   wcsspncpy(const wchar_t *, const wchar_t *);
15771 wchar_t  *wcsstr(const wchar_t *restrict, const wchar_t *restrict);
15772 double   wcstod(const wchar_t *restrict, wchar_t **restrict);
15773 float    wcstof(const wchar_t *restrict, wchar_t **restrict);
15774 wchar_t  *wcstok(wchar_t *restrict, const wchar_t *restrict,
15775         wchar_t **restrict);
15776 long     wcstol(const wchar_t *restrict, wchar_t **restrict, int);
15777 long double wcstold(const wchar_t *restrict, wchar_t **restrict);
15778 long long wcstoll(const wchar_t *restrict, wchar_t **restrict, int);
15779 unsigned long wcstoul(const wchar_t *restrict, wchar_t **restrict, int);
15780 unsigned long long wcstoull(const wchar_t *restrict, wchar_t **restrict, int);
15781
15782 XSI wchar_t  *wcsvcs(const wchar_t *, const wchar_t *);
15783 int     wcswidth(const wchar_t *, size_t);

```

```

15784     size_t      wcsxfrm(wchar_t *restrict, const wchar_t *restrict, size_t);
15785     int         wctob(wint_t);
15786     wctype_t    wctype(const char *);
15787 XSI    int         wcwidth(wchar_t);
15788     wchar_t     *wmemchr(const wchar_t *, wchar_t, size_t);
15789     int         wmemcmp(const wchar_t *, const wchar_t *, size_t);
15790     wchar_t     *wmemcpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15791     wchar_t     *wmemmove(wchar_t *, const wchar_t *, size_t);
15792     wchar_t     *wmemset(wchar_t *, wchar_t, size_t);
15793     int         wprintf(const wchar_t *restrict, ...);
15794     int         wscanf(const wchar_t *restrict, ...);

```

15795 The **<wchar.h>** header shall define the following macro names:

```

15796     WCHAR_MAX   The maximum value representable by an object of type wchar_t.
15797     WCHAR_MIN   The minimum value representable by an object of type wchar_t.
15798     WEOF        Constant expression of type wint_t that is returned by several WP functions
15799               to indicate end-of-file.
15800     NULL        As described in <stddef.h>.
15801     The tag tm shall be declared as naming an incomplete structure type, the contents of which are
15802               described in the header <time.h>.
15803     Inclusion of the <wchar.h> header may make visible all symbols from the headers <ctype.h>,
15804               <stdio.h>, <stdarg.h>, <stdlib.h>, <string.h>, <stddef.h>, and <time.h>.

```

15805 APPLICATION USAGE

15806 None.

15807 RATIONALE

15808 None.

15809 FUTURE DIRECTIONS

15810 None.

15811 SEE ALSO

15812 **<ctype.h>**, **<stdarg.h>**, **<stddef.h>**, **<stdio.h>**, **<stdlib.h>**, **<string.h>**, **<time.h>**, the System
15813 Interfaces volume of IEEE Std. 1003.1-200x, *btowc()*, *fgetwc()*, *fgetws()*, *fputwc()*, *fputws()*,
15814 *fwide()*, *fwprintf()*, *fwscanf()*, *getwc()*, *getwchar()*, *iswalnum()*, *iswalphalpha()*, *iswcntrl()*, *iswctype()*,
15815 *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*,
15816 *iswctype()*, *mbsinit()*, *mbrlen()*, *mbrtowc()*, *mbsrtowcs()*, *putwc()*, *putwchar()*, *swprintf()*, *swscanf()*,
15817 *towlower()*, *towupper()*, *ungetwc()*, *vfwprintf()*, *vfwscanf()*, *vswprintf()*, *vswscanf()*, *vscanf()*,
15818 *wcrtomb()*, *wcsrtombs()*, *wscat()*, *wcschr()*, *wscmp()*, *wscoll()*, *wscopy()*, *wscspn()*, *wcsftime()*,
15819 *wcslen()*, *wcsncat()*, *wcsncmp()*, *wcsncpy()*, *wcspbrk()*, *wcsrchr()*, *wcsspn()*, *wcsstr()*, *wcstod()*,
15820 *wcstof()*, *wcstok()*, *wcstol()*, *wcstold()*, *wcstoll()*, *wcstoul()*, *wcstoull()*, *wcswcs()*, *wcswidth()*,
15821 *wcsxfrm()*, *wctob()*, *wctype()*, *wcwidth()*, *wmemchr()*, *wmemcmp()*, *wmemcpy()*, *wmemmove()*,
15822 *wmemset()*, *wprintf()*, *wscanf()*

15823 CHANGE HISTORY

15824 First released in Issue 4.

15825 Issue 5

15826 Aligned with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

15827 **Issue 6**

15828 The Open Group corrigenda item U021/10 has been applied. The prototypes for *wcswidth()* and
15829 *wcwidth()* are marked as extensions.

15830 The Open Group corrigenda item U028/5 has been applied, correcting the prototype for the
15831 *mbsinit()* function.

15832 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 15833 • Various function prototypes are updated to add the **restrict** keyword.
- 15834 • The functions *vwscanf()*, *vswscanf()*, *wcstof()*, *wcstold()*, *wcstoll()*, and *wcstoull()* are added.

15835 **NAME**

15836 wctype.h — wide-character classification and mapping utilities

15837 **SYNOPSIS**

15838 #include <wctype.h>

15839 **DESCRIPTION**

15840 cx The functionality described on this reference page extends the ISO C standard. Applications
 15841 shall define the appropriate feature test macro (see the System Interfaces volume of
 15842 IEEE Std. 1003.1-200x, Section 2.2, The Compilation Environment) to enable the visibility of
 15843 symbols in this header.

15844 The **<wctype.h>** header shall define the following data types through **typedef**:15845 **wint_t** As described in **<wchar.h>**.15846 **wctrans_t** A scalar type that can hold values which represent locale-specific character
15847 mappings.15848 **wctype_t** As described in **<wchar.h>**.15849 The **<wctype.h>** header shall declare the following as functions and may also define them as
15850 macros. Function prototypes shall be provided for use with an ISO C standard compiler.

```
15851 int      iswalnum(wint_t);
15852 int      iswalpha(wint_t);
15853 int      iswblank(wint_t);
15854 int      iswcntrl(wint_t);
15855 int      iswdigit(wint_t);
15856 int      iswgraph(wint_t);
15857 int      iswlower(wint_t);
15858 int      iswprint(wint_t);
15859 int      iswpunct(wint_t);
15860 int      iswspace(wint_t);
15861 int      iswupper(wint_t);
15862 int      iswxdigit(wint_t);
15863 int      iswctype(wint_t, wctype_t);
15864 wint_t   towctrans(wint_t, wctrans_t);
15865 wint_t   towlower(wint_t);
15866 wint_t   towupper(wint_t);
15867 wctrans_t wctrans(const char *);
15868 wctype_t wctype(const char *);
```

15869 The **<wctype.h>** header shall define the following macro name:15870 **WEOF** Constant expression of type **wint_t** that is returned by several MSE functions
15871 to indicate end-of-file.

15872 For all functions described in this header that accept an argument of type **wint_t**, the value is
 15873 representable as a **wchar_t** or equals the value of **WEOF**. If this argument has any other value,
 15874 the behavior is undefined.

15875 The behavior of these functions shall be affected by the *LC_CTYPE* category of the current locale.15876 Inclusion of the **<wctype.h>** header may make visible all symbols from the headers **<ctype.h>**,
15877 **<stdio.h>**, **<stdarg.h>**, **<stdlib.h>**, **<string.h>**, **<stddef.h>**, **<time.h>**, and **<wchar.h>**.

15878 **APPLICATION USAGE**

15879 None.

15880 **RATIONALE**

15881 None.

15882 **FUTURE DIRECTIONS**

15883 None.

15884 **SEE ALSO**

15885 <locale.h>, <wchar.h>, the System Interfaces volume of IEEE Std. 1003.1-200x, *iswalnum()*,
15886 *iswalph()*, *iswblank()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,
15887 *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *towctrans()*, *towlower()*, *towupper()*,
15888 *wctrans()*, *wctype()*

15889 **CHANGE HISTORY**

15890 First released in Issue 5. Derived from the ISO/IEC 9899:1990/Amendment 1:1995 (E).

15891 **Issue 6**15892 The *iswblank()* function is added for alignment with the ISO/IEC 9899:1999 standard.

15893 **NAME**

15894 wordexp.h — word-expansion types

15895 **SYNOPSIS**

15896 #include <wordexp.h>

15897 **DESCRIPTION**15898 The **<wordexp.h>** header shall define the structures and symbolic constants used by the
15899 *wordexp()* and *wordfree()* functions.15900 The structure type **wordexp_t** shall contain at least the following members:

15901	size_t	we_wordc	Count of words matched by <i>words</i> .
15902	char	**we_wordv	Pointer to list of expanded words.
15903	size_t	we_offs	Slots to reserve at the beginning of <i>we_wordv</i> .

15904 The *flags* argument to the *wordexp()* function shall be the bitwise-inclusive OR of the following
15905 flags:

15906	WRDE_APPEND	Append words to those previously generated.
15907	WRDE_DOOFFS	Number of null pointers to prepend to <i>we_wordv</i> .
15908	WRDE_NOCMD	Fail if command substitution is requested.
15909	WRDE_REUSE	The <i>pwordexp</i> argument was passed to a previous successful call to 15910 <i>wordexp()</i> , and has not been passed to <i>wordfree()</i> . The result is the same 15911 as if the application had called <i>wordfree()</i> and then called <i>wordexp()</i> 15912 without WRDE_REUSE.
15913	WRDE_SHOWERR	Do not redirect <i>stderr</i> to /dev/null .
15914	WRDE_UNDEF	Report error on an attempt to expand an undefined shell variable.

15915 The following constants shall be defined as error return values:

15916	WRDE_BADCHAR	One of the unquoted characters—<newline>, ' ', ' & ', ' ; ', ' < ', ' > ', 15917 ' (', ') ', ' { ', ' } '—appears in <i>words</i> in an inappropriate context.
15918	WRDE_BADVAL	Reference to undefined shell variable when WRDE_UNDEF is set in <i>flags</i> .
15919	WRDE_CMDSUB	Command substitution requested when WRDE_NOCMD was set in <i>flags</i> .
15920	WRDE_NOSPACE	Attempt to allocate memory failed.
15921	WRDE_NOSYS	The implementation does not support the function.
15922	WRDE_SYNTAX	Shell syntax error, such as unbalanced parentheses or unterminated 15923 string.

15924 The following shall be declared as functions and may also be declared as macros. Function
15925 prototypes shall be provided for use with an ISO C standard compiler.

```
15926 int wordexp(const char *restrict, wordexp_t *restrict, int);
15927 void wordfree(wordexp_t *);
```

15928 The implementation may define additional macros or constants using names beginning with
15929 WRDE_.

15930 **APPLICATION USAGE**

15931 None.

15932 **RATIONALE**

15933 None.

15934 **FUTURE DIRECTIONS**

15935 None.

15936 **SEE ALSO**15937 The System Interfaces volume of IEEE Std. 1003.1-200x, *wordexp()*, the Shell and Utilities volume
15938 of IEEE Std. 1003.1-200x15939 **CHANGE HISTORY**

15940 First released in Issue 4. Derived from the ISO POSIX-2 standard.

15941 **Issue 6**15942 The **restrict** keyword is added to the prototype for *wordexp()*.

