

**The Open Group Base Working Group**

**New API Extensions (Extended Interfaces  
Strawman Draft 7.3)**

**Work Item # 1.2.3**

November 2004

© Copyright 2004, The Open Group

All rights reserved.

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

Boundaryless Information Flow™ is a trademark and UNIX® and The Open Group® are registered trademarks of The Open Group in the United States and other countries.

## **The Open Group**

Document Number: This draft document is work item 1.2.3 of The Open Group Base Working Group Work Plan dated 9 June 2004.

Comments relating to the material contained in this document may be submitted in aardvark format to:

The Open Group  
Thames Tower  
37-45 Station Road  
Reading  
Berkshire, RG1 1LX  
United Kingdom

<http://www.opengroup.org/platform/bugreport.html>

See <http://www.opengroup.org/austin/aardvark/format.html> for information on the aardvark comment format.

# New API Extensions (*Extended Interfaces Strawman draft 7.3*)

1

2       **Note:**       Except as permitted below, no part of this document may be reproduced, stored in a retrieval  
3                   system, or transmitted, in any form or by any means, electronic, mechanical, photocopying ,  
4                   recording or otherwise, without the prior permission of the copyright holders.

5                   This is an unapproved draft documents, subject to change. Permission is granted to Austin  
6                   Group participants to download and reproduce this document for the purposes of Austin  
7                   Group standardization activities. Other entities or persons seeking permission to reproduce  
8                   this documents, or to reproduce portions of the document for any purpose must contact the  
9                   copyright owner for express written permission. Use of information contained in these  
10                  unapproved drafts is at your own risk.

11                 The purpose of this document is to define a set of New API Extensions to further increase  
12                 application capture and hence portability for systems built upon version 3 of the Single UNIX  
13                 Specification .

14                 The scope of this set of extensions has been to consider interfaces from existing open source  
15                 implementations such as the GNU C library.

16                 No decision has been made on whether these interfaces will be added to a future technical  
17                 standard of The Open Group, how these interfaces would announce themselves in the  
18                 namespace, or whether related interfaces should be merged with existing pages.

## 19 1.1 Change History

### 20 Draft 7.3

- 21 • Remove *endusershell()*, *getusershell()*, *memmem()*, *on\_exit()*, and *setusershell()*.
- 22 • Add additional reviewers comments to further pages.

### 23 Draft 7.2

- 24 • Remove *alloca()*, *strdupa()* and *strndupa()*.
- 25 • Add additional reviewers comments to pages initially reviewed by the Base Working Group
- 26 • Fixup *on\_exit()* prototype in frontmatter.

### 27 Draft 7.1

- 28 • Reorder the *scandir()* manual page to list the *alphasort()* function first.
- 29 • Update `<string.h>` so that *strdupa()* and *strndupa()* are listed separately.
- 30 • Update the example in *open\_memstream()*.
- 31 • Update descriptions of *stpcpy()* and *stpncpy()* to be closer to *strcpy()*.
- 32 • Update *strdupa()* and *strndupa()* so that it can either be implemented as a function or a
- 33 macro. This is for consistency with *alloca()*.

### 34 Draft 7

35 Minor updates for proposal to have this set as part of an Extended Interfaces Option Group.

### 36 Draft 6

- 37 • Numerous updates after comments on draft 5.
- 38 • Added new functions *fnmemopen()* and *open\_memstream()*

### 39 Draft 5

40 Key changes in draft 5 are as follows

- 41 • Removed *hcreate\_r()*, *hdestroy\_r()*, and *hsearch\_r()*.
- 42 • Added EINVAL error to *dirfd()* and error return of `-1`.
- 43 • Added EBADF error to *dprintf()*.
- 44 • Removed *fgetgrent()*, *fgetgrent\_r()*, *fgetpwent()*, and *fgetpwent\_r()*.
- 45 • Merged *getdelim()* and *getline()* pages, corrected return types to *ssize\_t* and tidy up error
- 46 cases.
- 47 • Corrected DESCRIPTION of *mbsnrtowcs()* since *nmc* is the input buffer size in bytes and a
- 48 general rewrite more in line with *mbsrtowcs()*.
- 49 • *mkdtemp()* should have 6 Xs not 7, plus tidy up of the RETURN VALUE section.
- 50 • Updates to *on\_exit()* to make it clearer the interworking with *atexit()*.
- 51 • A rewrite of *scandir()*.

- 52 • Tidy up the RETURN VALUE in *stpncpy()* and *wcpncpy()*.
- 53 • Remove extraneous ENOMEM errors for *strdupa()* and *strndupa()*.
- 54 • Some tidy up to the DESCRIPTION of *strlen()* to make it clearer that only *maxlen* bytes are
- 55 examined.
- 56 • Correct the RETURN VALUE in *wcpncpy()* as per *stpncpy()*.
- 57 • A tidy up of *strsignal()*.
- 58 • Make it clear that its the first *nwc* wide characters for *wcsnrtombs()*.

## 59 1.2 XBD Changes

60 It is proposed that these additions comprise a new option group, called the Extended Interfaces  
61 option group.

### 62 1.2.1 1.5.1 Codes

63 Add a new margin marker code "EX Extended Interfaces", with the text

64 "The functionality described is optional. The functionality described is also an extension to the  
65 ISO C standard.

66 Where applicable, functions are marked with the EX margin legend in the SYNOPSIS section.  
67 Where additional semantics apply to a function, the material is identified by use of the EX  
68 margin legend."

### 69 1.2.2 13. Headers

70 The following header file man pages will need the following additions, margin marked and  
71 shaded as part of the EX option group.

#### 72 <dirent.h>

73 The following shall be declared as functions and may also be defined as macros. Function  
74 prototypes shall be provided.

```
75     int alphasort(const struct dirent **, const struct dirent **);
76     int dirfd (DIR *);
77     int scandir (const char *, struct dirent ***,
78                int (*) (const struct dirent *),
79                int (*) (const struct dirent **, const struct dirent **));
```

#### 80 <signal.h>

81 The following shall be declared as functions and may also be defined as macros. Function  
82 prototypes shall be provided.

```
83     void psignal (int, const char *);
```

#### 84 <stdio.h>

85 The following shall be declared as functions and may also be defined as macros. Function  
86 prototypes shall be provided.

```
87     int dprintf (int, const char *, ...);
88     FILE *fmemopen(void *, size_t, const char *);
89     ssize_t getdelim (char **, size_t *, int, FILE *);
90     ssize_t getline (char **, size_t *, FILE *);
91     FILE *open_memstream(char **, size_t *);
```

#### 92 <stdlib.h>

93 The following shall be declared as functions and may also be defined as macros. Function  
94 prototypes shall be provided.

```
95     char *mkdtemp(char *);
```

#### 96 <string.h>

97 The following shall be declared as functions and may also be defined as macros. Function  
98 prototypes shall be provided.

```
99     char *stpcpy (char *, const char *);
100     char *stpncpy (char *, const char *, size_t);
101     char *strndup (const char *, size_t);
102     size_t strnlen (const char *, va_list);
103     char *strsignal(int signum);
```

104 **<wchar.h>**

105 The following shall be declared as functions and may also be defined as macros. Function  
106 prototypes shall be provided.

```
107     size_t mbsnrtowcs (wchar_t *, const char **, size_t, size_t, mbstate_t *);
108     wchar_t *wcpcpy (wchar_t *, const wchar_t *);
109     wchar_t *wcpncpy (wchar_t *, const wchar_t *, size_t);
110     int wcscasecmp (const wchar_t *, const wchar_t *);
111     wchar_t *wcsdup (const wchar_t *);
112     int wcsncasecmp (const wchar_t *, const wchar_t *, size_t);
113     size_t wcsnlen (const wchar_t *, size_t);
114     size_t wcsnrtombs (char *, const wchar_t **, size_t, size_t, mbstate_t *);
```

115 **1.3 XSH Manual Pages**

116 The man pages follow.



117 **NAME**

118       alphasort, scandir — scan a directory

119 **SYNOPSIS**

```
120     #include <dirent.h>
121
122     int alphasort(const struct dirent **d1, const struct dirent **d2);
123
124     int scandir(const char *dir, struct dirent ***namelist,
125                int (*sel)(const struct dirent *),
126                int (*compar)(const struct dirent **, const struct dirent **));
```

126 **DESCRIPTION**

127       The *alphasort()* function can be used as the comparison function for the *scandir()* function to  
 128       sort the directory entries into alphabetical order, as if by the *strcoll()* function. Its parameters  
 129       are the two directory entries, *d1* and *d2*, to compare.

130       The *scandir()* function shall scan the directory *dir*, calling the *sel()* function on each directory  
 131       entry. Entries for which *sel()* returns non-zero shall be stored in strings allocated via *malloc()*,  
 132       and sorted using *qsort()* with the comparison function *compar()*, and collected in array *namelist*  
 133       which shall be allocated via *malloc()*. If *sel()* is a null pointer, all entries shall be selected.

134 **RETURN VALUE**

135       Upon successful completion, *alphasort()* shall return an integer greater than, equal to, or less  
 136       than 0, according to whether the directory pointed to by *d1* is greater than, equal to, or less than  
 137       the directory pointed to by *d2* when both are interpreted as appropriate to the current locale.  
 138       There is no return value reserved to indicate an error.

139       Upon successful completion the *scandir()* function shall return the number of entries in the  
 140       array and a pointer to the array through the parameter *namelist*. The *scandir()* function shall  
 141       return  $-1$  if the directory cannot be opened for reading or if *malloc()* cannot allocate enough  
 142       memory to hold all the data structures.

143 **ERRORS**

144       The *scandir()* function shall fail if:

145       [EACCES]       Search permission is denied for the component of the path prefix of *dir* or read  
 146       permission is denied for *dir*.

147       [ELOOP]        A loop exists in symbolic links encountered during resolution of the *dir*  
 148       argument.

149       [ENAMETOOLONG]   The length of the *dir* argument exceeds {PATH\_MAX} or a pathname  
 150       component is longer than {NAME\_MAX}.

151       [ENOENT]        A component of *dir* does not name an existing directory or *dir* is an empty  
 152       string.

153       [ENOMEM]        Insufficient storage space is available.

154       [ENOTDIR]       A component of *dir* is not a directory.

155       The *scandir()* function may fail if:

156       [ELOOP]        More than {SYMLOOP\_MAX} symbolic links were encountered during  
 157       resolution of the *dir* argument.

158       [EMFILE]        {OPEN\_MAX} file descriptors are currently open in the calling process.

160 [ENAMETOOLONG]  
161 As a result of encountering a symbolic link in resolution of the *dir* argument,  
162 the length of the substituted pathname string exceeded {PATH\_MAX}.

163 [ENFILE] Too many files are currently open in the system.

#### 164 EXAMPLES

165 An example that print the files in the current directory:

```
166 #include <dirent.h>
167 #include <stdio.h>
168 ...
169 struct dirent **namelist;
170 int i,n;

171     n = scandir(".", &namelist, 0, alphasort);
172     if (n < 0)
173         perror("scandir");
174     else {
175         for (i = 0; i < n; i++) {
176             printf("%s\n", namelist[i]->d_name);
177             free(namelist[i]);
178         }
179         free(namelist);
180     ...
```

#### 181 APPLICATION USAGE

182 These functions are part of the Extended Interfaces option and need not be available on all  
183 implementations.

#### 184 RATIONALE

185 None.

#### 186 FUTURE DIRECTIONS

187 None.

#### 188 SEE ALSO

189 the Base Definitions volume of IEEE Std 1003.1-2001, <**dirent.h**>

#### 190 CHANGE HISTORY

191 First released in Issue X

192 **NAME**

193 dirfd — extracts the file descriptor used by a DIR stream

194 **SYNOPSIS**

```
195 #include <dirent.h>
196 int dirfd(DIR *dirp);
197
```

198 **DESCRIPTION**199 **Notes to Reviewers**200 *This section with side shading will not appear in the final copy. - Ed.*201 **Commentary on this function:**

202 This interface was introduced because glibc does not make public the **DIR** data structure.  
 203 Applications tend to use the *fhdir()* function on the file descriptor returned by this interface,  
 204 and this has proven useful for security reasons, in particular it is a better technique than others  
 205 where directory names might change. The working group has some concern that a file  
 206 descriptor is not required for the **DIR** data structure in the present standard, so there would be a  
 207 need either to prefix *dirfd()*, with text along the lines of "If a file descriptor is used to  
 208 implement...", or to require an underlying file descriptor. The former would require applications  
 209 to know about the implementation, and hence applications would not be able to make portable  
 210 use of this function.

211 Thus the implication would be that to introduce this we would have to mandate an underlying  
 212 file descriptor for a DIR object for it to be useful for portable applications.

213 So if we take this change it would need a number of other changes to the existing directory  
 214 related functions.

215 The *dirfd()* function shall return the file descriptor used by the *dirp* argument.

216 **RETURN VALUE**

217 Upon successful completion, the *dirfd()* function shall return an integer which contains the file  
 218 descriptor for the stream pointed to by *dirp*. Otherwise it shall return -1 and may set *errno* to  
 219 indicate the error.

220 **ERRORS**

221 The *dirfd()* function may fail if:

222 [EINVAL] The *dirp* argument does not refer to a valid directory stream.

223 **EXAMPLES**

224 None.

225 **APPLICATION USAGE**

226 The *dirfd()* function is part of the Extended Interfaces option and need not be available on all  
 227 implementations.

228 **RATIONALE**

229 None.

230 **FUTURE DIRECTIONS**

231 None.

232 **SEE ALSO**233        *opendir()* the Base Definitions volume of IEEE Std 1003.1-2001, <**dirent.h**>234 **CHANGE HISTORY**

235        First released in Issue X

236 **NAME**237 `dprintf` — formatted output conversion to a file descriptor238 **SYNOPSIS**239 `#include <stdio.h>`240 `int dprintf(int fildes, const char *format, ...);`

241

242 **DESCRIPTION**243 **Notes to Reviewers**244 *This section with side shading will not appear in the final copy. - Ed.*245 **Commentary on this function:**

246 It is unclear as to what the required buffering behavior is for this function. More information is  
 247 needed. On the surface this function would appear to be a convenience function more than a  
 248 necessity. Is this really done frequently enough to justify adding a new function when *snprintf()*  
 249 and *write()* are sufficient to do the job? It was also suggested that *fdprintf()* would be a better  
 250 name.

251 The *dprintf()* function shall be equivalent to the *printf()* function, producing output according to  
 252 the contents of *format*, with the exception that instead of the output going to *stdout*, the output of  
 253 *dprintf()* is directed to the file descriptor *fildes*.

254 **RETURN VALUE**255 Upon successful completion, the *dprintf()* function shall return the number of bytes transmitted.

256 If an output error was encountered, it shall return a negative value.

257 **ERRORS**258 Refer to *fprintf()*.259 In addition, the *dprintf()* function may fail if:260 [EBADF] The *fildes* argument is not a valid file descriptor.261 **EXAMPLES**

262 None.

263 **APPLICATION USAGE**

264 The *dprintf()* function is part of the Extended Interfaces option and need not be available on all  
 265 implementations.

266 **RATIONALE**

267 None.

268 **FUTURE DIRECTIONS**

269 None.

270 **SEE ALSO**271 *printf()* the Base Definitions volume of IEEE Std 1003.1-2001, `<stdio.h>`272 **CHANGE HISTORY**

273 First released in Issue X

274 **NAME**

275 fmemopen — open a memory buffer stream

276 **SYNOPSIS**

277 #include &lt;stdio.h&gt;

278 FILE \*fmemopen(void \*restrict buf, size\_t size, const char \*restrict mode);

279

280 **DESCRIPTION**281 **Notes to Reviewers**282 *This section with side shading will not appear in the final copy. - Ed.*283 **Commentary on this function:**

284 This interface has been introduced to eliminate many of the errors encountered in the  
 285 construction of strings, notably overflowing of strings. This interface prevents overflow. A wide  
 286 character version has not yet been proposed. It was proposed that *fmemopen()* should leave the  
 287 results unoriented.

288 There appears to be a need to modify other related stdio pages that talk about handling **FILE**  
 289 objects; how would they behave if a memory stream is underlying the stream? If writes on a  
 290 stream with an underlying memory buffer, would overflow the memory buffer, the behavior is  
 291 as the same as filesystem full, that is [ENOSPC].

292 Further work would be needed to cleanup this page, and other pages.

293 The *fmemopen()* function shall associate the buffer given by the *buf* and *size* arguments with a  
 294 stream. The *buf* argument shall be either a null pointer or point to a buffer that is at least *size*  
 295 bytes long.

296 The *mode* argument is a character string having one of the following values:

297	<i>r</i> or <i>rb</i>	Open the stream for reading.
298	<i>w</i> or <i>wb</i>	Open the stream for writing.
299	<i>a</i> or <i>ab</i>	Append; open the stream for writing at the first null byte.
300	<i>r+</i> or <i>rb+</i> or <i>r+b</i>	Open the stream for update (reading and writing).
301	<i>w+</i> or <i>wb+</i> or <i>w+b</i>	Open the stream for update (reading and writing). Truncate the buffer 302 contents.
303	<i>a+</i> or <i>ab+</i> or <i>a+b</i>	Append; open the stream for update (reading and writing); the initial 304 position is at the first null byte.

305 The character 'b' shall have no effect, but is allowed for ISO C standard conformance.

306 If a null pointer is specified as the *buf* argument, *fmemopen()* shall use *malloc()* to allocate a buffer  
 307 that is *size* bytes long. This buffer shall be automatically freed when the stream is closed. Because  
 308 this feature is only useful when the stream is opened for updating (because there is no way to  
 309 get a pointer to the buffer) the *fmemopen()* call may fail if the *mode* argument does not include a  
 310 '+'.  
 311

312 The stream maintains a current position in the buffer. This position is initially set to either the  
 313 begin of the buffer (for *r* and *w* modes) or to the first null byte in the buffer (for *a* modes). If no  
 null byte is found in append mode, the initial position is set to one byte after the end of the

314 buffer.

315 The stream also maintains the size of the current buffer contents. For modes *r* and *r+* the size is  
 316 set to the value given by the *size* argument. For modes *w* and *w+* the initial size is zero and for  
 317 modes *a* and *a+* the initial size is either the position of the first null byte in the buffer or the value  
 318 of the *size* argument if no null byte is found.

319 A read operation on the stream cannot advance the current buffer position behind the current  
 320 buffer size. Reaching the buffer size in a read operation counts as "end of file". Null bytes in the  
 321 buffer have no special meaning for reads. The write operation starts at the current buffer  
 322 position of the stream.

323 A write operation starts either at the current position of the stream (if mode has not specified a  
 324 as the first character) or at the current size of the stream (if mode had *a* as the first character). If  
 325 the current position at the end of the write is larger than the current buffer size, the current  
 326 buffer size is set to the current position. A write operation on the stream cannot advance the  
 327 current buffer size behind the size given in the *size* argument.

328 When a stream open for writing is flushed or closed, a null byte is written at the end of the buffer  
 329 if it fits. If a stream open for update is flushed or closed and the last write has advanced the  
 330 current buffer size, a null byte is written at the end of the buffer if it fits.

331 An attempt to seek a memory buffer stream to a negative position or to a position larger than the  
 332 buffer size given in the *size* argument shall fail.

### 333 RETURN VALUE

334 Upon successful completion, *fmemopen()* shall return a pointer to the object controlling the  
 335 stream. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

### 336 ERRORS

337 The *fmemopen()* function shall fail if:

338 [EINVAL] The *size* argument specifies a buffer size of zero.

339 The *fmemopen()* function may fail if:

340 [EINVAL] The value of the *mode* argument is not valid.

341 [EINVAL] The *buf* argument is a null pointer and the *mode* argument does not include a  
 342 '+' character.

343 [ENOMEM] The *buf* argument is a null pointer and the allocation of a buffer of length *size*  
 344 has failed.

345 [EMFILE] {FOPEN\_MAX} streams are currently open in the calling process.

### 346 EXAMPLES

```
347 #include <stdio.h>
348
349 static char buffer[] = "foobar";
350
351 int
352 main (void)
353 {
354     int ch;
355     FILE *stream;
356
357     stream = fmemopen(buffer, strlen (buffer), "r");
358     if (stream == NULL)
```

```
356         /* handle error */;
357     while ((ch = fgetc(stream)) != EOF)
358         printf("Got %c\n", ch);
359
359     fclose(stream);
360     return (0);
361 }
```

362 This program produces the following output:

```
363         Got f
364         Got o
365         Got o
366         Got b
367         Got a
368         Got r
```

#### 369 **APPLICATION USAGE**

370 The *fmemopen()* function is part of the Extended Interfaces option and need not be available on  
371 all implementations.

#### 372 **RATIONALE**

373 None.

#### 374 **FUTURE DIRECTIONS**

375 None.

#### 376 **SEE ALSO**

377 *fdopen()*, *fopen()*, *freopen()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**stdio.h**>

#### 378 **CHANGE HISTORY**

379 First released in Issue X



380 **NAME**381 getdelim — reads a delimited record from *stream*.382 **SYNOPSIS**

383 #include &lt;stdio.h&gt;

384 ssize\_t getdelim(char \*\*lineptr, size\_t \*n, int delimiter,  
385 FILE \*stream);386 ssize\_t getline(char \*\*lineptr, size\_t \*n, FILE \*stream);  
387388 **DESCRIPTION**389 **Notes to Reviewers**390 *This section with side shading will not appear in the final copy. - Ed.*391 **Commentary on this function:**392 These functions are widely used to solve the problem that the *fgets()* function has with long  
393 lines. The functions automatically enlarge the target buffers if needed. These are especially  
394 useful since they reduce code needed for applications.395 More words needed on the description, need to clean up bytes vs characters to be compatible  
396 with the standard397 The *getdelim()* function shall read from *stream* until it encounters a byte matching the *delimiter*  
398 character. The argument *delimiter* (when converted to a **char**) shall specify the character that  
399 terminates the read process.400 The *delimiter* argument is an **int**, the value of which the application shall ensure is a character  
401 representable as an **unsigned char** or equal value to the macro EOF. If the *delimiter* argument  
402 has any other value, the behavior is undefined.403 The application shall ensure that *\*lineptr* is a valid argument that could be passed to the *free()*  
404 function. If *\*n* is nonzero, the application shall ensure that *\*lineptr* points to an object containing  
405 at least *\*n* bytes.406 The size of the object pointed to by *\*lineptr* shall be increased to fit the incoming line, if it isn't  
407 already large enough. The bytes read shall be stored in the string pointed to by the argument  
408 *lineptr*.409 The *getline()* function shall be equivalent to the *getdelim()* function with the *delimiter* character  
410 equal to the newline character.411 **RETURN VALUE**412 Upon successful completion the *getdelim()* function shall return the number of bytes written into  
413 the buffer, including the delimiter character if one was encountered before EOF. Otherwise it  
414 shall return -1 and set *errno* to indicate the error.415 **ERRORS**416 The *getdelim()* and *getline()* functions shall fail if:417 [EINVAL] When *lineptr* or *n* are a null pointer.418 The *getdelim()* and *getline()* functions may fail if:419 [EINVAL] *stream* is not a valid file descriptor.

420 [Eoverflow] More than Ssize\_max bytes were read without encountering the *delimiter*  
421 character.

**422 EXAMPLES****423 APPLICATION USAGE**

424 The *getdelim()* and *getline()* functions are part of the Extended Interfaces option and need not be  
425 available on all implementations.

426 Setting *\*lineptr* to a null pointer and *\*n* to zero are allowed and a recommended way to start  
427 parsing a file.

**428 RATIONALE**

429 These functions have been explicitly designed to enlarge the buffer if necessary.

**430 FUTURE DIRECTIONS**

431 None.

**432 SEE ALSO**

433 the Base Definitions volume of IEEE Std 1003.1-2001, <**stdio.h**>

**434 CHANGE HISTORY**

435 First released in Issue X

436 **NAME**437 `mbsnrtowcs` — converts a multi-byte string to a wide character string.438 **SYNOPSIS**439 

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

440 

```
size_t mbsnrtowcs(wchar_t *restrict dst, const char **restrict src,  
441 size_t nmc, size_t len, mbstate_t *restrict ps);
```

442

443 **DESCRIPTION**444 The `mbsnrtowcs()` function works like the `mbsrtowcs()` function, except that the conversion of  
445 characters pointed to by `src` is limited to at most `nmc` bytes (the size of the input buffer).446 If `dst` is not a null pointer, then `mbsnrtowcs()` shall attempt to convert `nmc` bytes from the multi  
447 byte string pointed to by `src` into a wide character string starting at `dst`. No more than `len` wide  
448 characters shall be written to `dst`. The shift state, pointed at by `ps` is updated by the conversion.  
449 Each conversion shall take place, as if by repeated calls to `mbrtowc(dst, *src, n, ps)` where `n` is a  
450 positive number. As long as this call succeeds, it is repeated, each time incrementing `dst` by one  
451 and `*src` by the number of bytes converted.

452 Conversion shall stop early if any of the following cases occurs:

453 1. An invalid sequence of bytes was encountered in the `src` buffer. Under these conditions `*src` is  
454 left pointing to the bytes which caused the conversion to halt. `-1` is returned, and `errno` is set to  
455 `EILSEQ`.456 2. Either the `nmc` limit has been reached, or `len` non-null wide characters have already been  
457 stored in `dst`. Here, `*src` is left to point to the next multi byte sequence that has not been  
458 converted, and the total number of wide characters written to `dst` is returned.459 3. The conversion of the multi byte buffer pointed to by `src` has been completed by encountering  
460 a null byte. In this case `*src` is set to a null pointer, `*ps` is returned to its initial state, and the  
461 number of wide characters written to `dst`, excluding the terminating null character, is returned.462 When `dst` is a null pointer, the conversion proceeds as above, except that no wide characters are  
463 written to memory, and the `len` argument is ignored, so no destination length limit is imposed.464 In either case, if `ps` is a null pointer, `mbsnrtowcs()` shall use its own internal `mbstate_t` object,  
465 which is initialized at program start-up to the initial conversion state. Otherwise, the `mbstate_t`  
466 object pointed to by `ps` shall be used to completely describe the current conversion state of the  
467 associated character sequence.468 It is the responsibility of the calling program to ensure that `dst` is large enough to hold at least `len`  
469 wide characters.470 **RETURN VALUE**471 The `mbsnrtowcs()` function shall return the number of characters successfully converted, not  
472 including the terminating null (if any). If an error occurs, `mbsnrtowcs()` shall return `-1` and may  
473 set `errno`.474 **ERRORS**475 The `mbsnrtowcs()` function may fail if:476 

[EILSEQ]	An invalid multi byte sequence was encountered.
----------	---

477 **EXAMPLES**

478       None.

479 **APPLICATION USAGE**480       The *mbsnrto wcs()* function is part of the Extended Interfaces option and need not be available on  
481       all implementations.482 **RATIONALE**

483       None.

484 **FUTURE DIRECTIONS**

485       None.

486 **SEE ALSO**487       *mbsrtowcs()*, *iconv()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**wchar.h**>488 **CHANGE HISTORY**

489       First released in Issue X

490 **NAME**

491 mkdtemp — create a unique directory

492 **SYNOPSIS**

493 #include &lt;stdlib.h&gt;

494 char \*mkdtemp(char \*template);

495

496 **DESCRIPTION**

497 The *mkdtemp()* function uses the contents of *template* to construct a unique directory name. The  
 498 string provided in *template* shall be a filename ending with six trailing 'X's. The *mkdtemp()*  
 499 function shall replace each 'X' with a character from the portable filename character set. The  
 500 characters are chosen such that the resulting name does not duplicate the name of an existing  
 501 file at the time of a call to *mkdtemp()*. The unique directory name is used to attempt to create the  
 502 directory using mode 0700 as modified by the file creation mask.

503 **RETURN VALUE**

504 Upon successful completion the *mkdtemp()* function shall return a pointer to the string  
 505 containing the directory name if it was created. Otherwise it shall return a null pointer and set  
 506 *errno*.

507 **ERRORS**508 The *mkdtemp()* function shall fail if:

509 [EACCES] Search permission is denied on a component of the path prefix, or write  
 510 permission is denied on the parent directory of the directory to be created.

511 [EINVAL] The string pointed to by *template* does not end in 'XXXXXX'.

512 [ELOOP] A loop exists in symbolic links encountered during resolution of the path of  
 513 the directory to be created.

514 [EMLINK] The link count of the parent directory would exceed {LINK\_MAX}.

515 [ENAMETOOLONG] The length of the template argument exceeds {PATH\_MAX} or a  
 516 pathname component is longer than {NAME\_MAX}.

517 [ENOENT] A component of the path prefix specified by the template argument does not  
 518 name an existing directory or path is an empty string.

519 [ENOSPC] The file system does not contain enough space to hold the contents of the new  
 520 directory or to extend the parent directory of the new directory.

521 [ENOTDIR] A component of the path prefix is not a directory.

522 [EROFS] The parent directory resides on a read-only file system.

523 The *mkdtemp()* function may fail if:

524 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 525 resolution of the path of the directory to be created.

526 [ENAMETOOLONG] As a result of encountering a symbolic link in resolution of the path of the  
 527 directory to be created, the length of the substituted pathname string  
 528 exceeded {PATH\_MAX}.

529 **EXAMPLES**

530       None.

531 **APPLICATION USAGE**532       The *mkdtemp()* function is part of the Extended Interfaces option and need not be available on all  
533       implementations.534 **RATIONALE**

535       None.

536 **FUTURE DIRECTIONS**

537       None.

538 **SEE ALSO**539       *mkdir()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**stdlib.h**>540 **CHANGE HISTORY**

541       First released in Issue X

542 **NAME**

543 open\_memstream — open a dynamic memory buffer stream

544 **SYNOPSIS**

545 #include &lt;stdio.h&gt;

546 FILE \*open\_memstream(char \*\*bufp, size\_t \*sizep);

547

548 **DESCRIPTION**549 **Notes to Reviewers**550 *This section with side shading will not appear in the final copy. - Ed.*551 **Commentary on this function:**

552 This function is similar to *fmem\_open()* except that the memory is allocated dynamically by the  
 553 function. This would need a wide version *open\_wmemstream()*. This interface does not have a  
 554 *mode* parameter since it can only be written to.

555 It was agreed that further cleanup is needed on the wording.

556 The *open\_memstream()* function shall create a stream that is associated with a dynamically  
 557 allocated buffer. The buffer is obtained by calls to *malloc()* and *realloc()* and expanded as  
 558 necessary. It must be freed by the caller after closing the stream. The stream is opened for  
 559 writing and shall be seekable.

560 The stream maintains a current position in the allocated buffer and a current buffer length. The  
 561 position is initially set to zero (the begin of the buffer). Each write starts at the current position  
 562 and moves this position by the number of successfully written bytes. The length is initially set  
 563 to zero. If a write moves the position to a value larger than the current length, the current length  
 564 is set to this position. In this case a null byte is appended to the current buffer (but not accounted  
 565 for in the buffer length).

566 The maximum value of the buffer length and position is given by the smaller of {SIZE\_MAX}  
 567 and the maximum allowed file offset {OFF\_MAX}.

568 After a successful *flush()* or *fclose()* the locations pointed to by *bufp* and *sizep* contain the  
 569 address of the buffer and the current buffer length and the buffer is guaranteed to be terminated  
 570 by a null byte (which is not accounted for in the length).

571 An attempt to seek a dynamic buffer stream to a negative position or to a position larger than  
 572 the minimum of {SIZE\_MAX} and {OFF\_MAX} shall return an error.

573 **RETURN VALUE**

574 Upon successful completion, *open\_memstream()* shall return a pointer to the object controlling  
 575 the stream. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the  
 576 error.

577 **ERRORS**

578 The *open\_memstream()* function may fail if:

579 [EINVAL] *bufp* or *sizep* are NULL.

580 [EMFILE] {FOPEN\_MAX} streams are currently open in the calling process.

581 [ENOMEM] Memory for the stream or the buffer could not be allocated.

582 **EXAMPLES**

```
583     #include <stdio.h>
584     int main (void)
585     {
586
587         FILE *stream;
588         char *buf;
589         size_t len;
590
591         stream = fmemopen(&buf, &len);
592
593         if (stream == NULL)
594             /* handle error */;
595
596         fprintf(stream, "hello my world");
597         fflush(stream);
598         printf("buf=%s, len=%zu\n", buf, len);
599         fseeko(stream, 0, SEEK_SET);
600         fprintf(stream, "good-bye");
601         fclose(stream);
602         printf("buf=%s, len=%zu\n", buf, len);
603         free(buf);
604         return 0;
605     }
```

602 This program produces the following output:

```
603     buf=hello my world, len=14
604     buf=good-bye world, len=14
```

605 **APPLICATION USAGE**

606 The *open\_memstream()* function is part of the Extended Interfaces option and need not be  
607 available on all implementations.

608 **RATIONALE**

609 None.

610 **FUTURE DIRECTIONS**

611 None.

612 **SEE ALSO**

613 *fdopen()*, *fopen()*, *fmemopen()*, *freopen()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
614 **<stdio.h>**

615 **CHANGE HISTORY**

616 First released in Issue X



617 **NAME**

618       psignal — print signal information to standard error

619 **SYNOPSIS**

620       #include &lt;signal.h&gt;

621       void psignal(int *signum*, const char \**message*);

622

623 **DESCRIPTION**624 **Notes to Reviewers**625       *This section with side shading will not appear in the final copy. - Ed.*626       **Commentary on this function:**627       System V historically has *psignal()* and *psiginfo()* in <**siginfo.h**>.628       It was agreed during the preliminary review that there should be an additional *psiginfo()*  
629       function added since we have the type **siginfo\_t** within the standard.

630       The issue of which header the function(s) occur in needs to be resolved.

631       The *psignal()* function shall print a message out on *stderr* associated with a signal number. If  
632       *message* is not null and is not the empty string, then the string pointed to by the *message*  
633       argument shall be printed first, followed by a colon, a space, and the signal description string  
634       indicated by *signum*. If the *message* argument is null or points to an empty string, then only the  
635       signal description shall be printed. If *signum* is not a valid signal number, the behavior is  
636       implementation-defined.637 **RETURN VALUE**638       The *psignal()* function shall not return a value.639 **ERRORS**

640       No errors are defined.

641 **EXAMPLES**

642       None.

643 **APPLICATION USAGE**644       The *psignal()* function is part of the Extended Interfaces option and need not be available on all  
645       implementations.646 **RATIONALE**

647       None.

648 **FUTURE DIRECTIONS**

649       None.

650 **SEE ALSO**651       *perror()*, *strsignal()*, the Base Definitions volume of IEEE Std 1003.1-2001, <**signal.h**>652 **CHANGE HISTORY**

653       First released in Issue X

654 **NAME**

655 stpcpy — copy a string and return a pointer to the end of the result

656 **SYNOPSIS**

657 #include &lt;string.h&gt;

658 char \*stpcpy(char \*restrict dst, const char \*restrict src);

659

660 **DESCRIPTION**

661 The *stpcpy()* function shall be equivalent to *strcpy()*, copying the string pointed to by *src* into the  
662 array pointed to by *dst*, with the exception that *stpcpy()* shall return a pointer to the terminating  
663 null byte in *dst*, rather than the beginning of this array, allowing succeeding calls to add  
664 additional text to the *dst* array.

665 If copying takes place between objects that overlap, the behavior is undefined.

666 **RETURN VALUE**

667 The *stpcpy()* function shall return a pointer to the terminating null byte at the end of the *dst*  
668 buffer. No return values are reserved to indicate an error.

669 **ERRORS**

670 No errors are defined.

671 **EXAMPLES**

672 The following example demonstrates the construction of a multi part message in a single buffer.

673 #include &lt;string.h&gt;

674 #include &lt;stdio.h&gt;

675 int

676 main (void)

677 {

678 char buffer [10];

679 chsr \*name = buffer;

680 name = stpcpy (stpcpy (stpcpy (name, "ice"), "-"), "cream");

681 puts (buffer);

682 return 0;

683 }

684 **APPLICATION USAGE**

685 The *stpcpy()* function is part of the Extended Interfaces option and need not be available on all  
686 implementations.

687 **RATIONALE**

688 None.

689 **FUTURE DIRECTIONS**

690 None.

691 **SEE ALSO**692 *strcpy()*, the Base Definitions volume of IEEE Std 1003.1-2001, <string.h>693 **CHANGE HISTORY**

694 First released in Issue X

695 **NAME**

696        stpncpy — copy fixed length string, returning a pointer to the array end

697 **SYNOPSIS**

698        #include &lt;string.h&gt;

699        char \*stpncpy(char \*restrict *dst*, const char \*restrict *src*, size\_t *size*);

700

701 **DESCRIPTION**702 **Notes to Reviewers**703        *This section with side shading will not appear in the final copy. - Ed.*704        **Commentary on this function:**705        The 2nd paragraph of the DESCRIPTION is ambiguous (length of the string is usually  
706        equivalent to strlen(string), but it is off by 1 in this case) and needs to be fixed up.707        The *stpncpy()* function shall be equivalent to the *strcpy()* function, with the added restriction  
708        that it shall copy at most *size* bytes from *src* into *dst*.709        If *size* is smaller than the length of the string pointed to by *src* then no termination null byte shall  
710        be inserted into the *dst* array after the *size* bytes have been copied.711        If *size* is larger than the length of the string pointed to by *src* then all of the bytes in *src* are copied  
712        into the *dst* array. As many terminating null bytes are inserted as are needed to bring the total  
713        bytes transferred equal to *size*.

714        If copying takes place between objects that overlap, the behavior is undefined.

715 **RETURN VALUE**716        If a null byte is written to the destination, the *stpncpy()* function shall return the address of the  
717        first such null byte. Otherwise it shall return *src[size]*. No return values are reserved to indicate  
718        an error.719 **ERRORS**

720        No errors are defined.

721 **EXAMPLES**722 **APPLICATION USAGE**723        The *stpncpy()* function is part of the Extended Interfaces option and need not be available on all  
724        implementations.725        Applications must provide the space in *dst* for the *size* bytes to be transferred, as well as ensure  
726        that the *src* and *dst* array do not overlap.727 **RATIONALE**

728        None.

729 **FUTURE DIRECTIONS**

730        None.

731 **SEE ALSO**732        *strncpy()*, *strcpy()*, the Base Definitions volume of IEEE Std 1003.1-2001, <string.h>

733 **CHANGE HISTORY**  
734       First released in Issue X

735 **NAME**736 `strndup` — duplicate a specific number of bytes from a string737 **SYNOPSIS**738 `#include <string.h>`739 `char *strndup(const char *string, size_t size);`

740

741 **DESCRIPTION**

742 The `strndup()` function shall be equivalent to the `strdup()` function, duplicating the provided  
743 `string` in a new block of memory allocated using `malloc()`, with the exception being that `strndup()`  
744 copies at most `size` plus one bytes into the newly allocated memory, terminating the new string  
745 with a null byte.

746 If the length of `string` is larger than `size`, only `size` bytes shall be duplicated. If `size` is larger than  
747 the length of `string`, all bytes in `string` shall be copied into the new memory buffer, including the  
748 terminating null byte. The newly created string shall always be properly terminated.

749 **RETURN VALUE**

750 Upon successful completion the `strndup()` function shall return a pointer to the newly allocated  
751 memory containing the duplicated string. Otherwise it shall return a null pointer and set `errno` to  
752 indicate the error.

753 **ERRORS**754 The `strndup()` function shall fail if:

755 [ENOMEM] insufficient memory available for the target string.

756 **EXAMPLES**757 **APPLICATION USAGE**

758 The `strndup()` function is part of the Extended Interfaces option and need not be available on all  
759 implementations.

760 **RATIONALE**

761 None.

762 **FUTURE DIRECTIONS**

763 None.

764 **SEE ALSO**765 `strdup()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<string.h>`766 **CHANGE HISTORY**

767 First released in Issue X

768 **NAME**

769 strnlen — determine length of fixed size string

770 **SYNOPSIS**771 `#include <string.h>`772 `size_t strnlen(const char *s, size_t maxlen);`

773

774 **DESCRIPTION**775 **Notes to Reviewers**776 *This section with side shading will not appear in the final copy. - Ed.*777 **Commentary on this function:**778 The RETURN VALUE section wording is ambiguous. (How is "size of the string" related to  
779 string length?) Do we assume that the return value is strlen (s) or maxlen whichever is smaller?780 The *strnlen()* function shall compute the smaller of the number of bytes in the string to which *s*  
781 points not including the terminating null byte, or the value of the *maxlen* argument. The  
782 *strnlen()* function shall never examine more than *maxlen* bytes of the string pointed to by *s*.783 **RETURN VALUE**784 The *strnlen()* function shall return an integer containing the smaller of either the size of the  
785 string pointed to by *s* or *maxlen*.786 **ERRORS**

787 No errors are defined.

788 **EXAMPLES**789 **APPLICATION USAGE**790 The *strnlen()* function is part of the Extended Interfaces option and need not be available on all  
791 implementations.792 **RATIONALE**

793 None.

794 **FUTURE DIRECTIONS**

795 None.

796 **SEE ALSO**797 *strlen()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**798 **CHANGE HISTORY**

799 First released in Issue X

800 **NAME**

801 strsignal — get name of signal

802 **SYNOPSIS**

803 #include &lt;string.h&gt;

804 char \*strsignal(int *signum*);

805

806 **DESCRIPTION**807 **Notes to Reviewers**808 *This section with side shading will not appear in the final copy. - Ed.*809 **Commentary on this function:**810 Some implementations return NULL rather than unknown, so need to decide whether its worth  
811 picking one, perhaps unspecified is the best we can do for this interface.812 The *strsignal()* function shall map the signal number in *signum* to a implementation-defined  
813 string and shall return a pointer to it. It shall use the same set of messages as the *psignal()*  
814 function.815 The string pointed to shall not be modified by the application, but may be overwritten by a  
816 subsequent call to *strsignal()* or *setlocale()*.817 The contents of the message strings returned by *strsignal()* should be determined by the setting  
818 of the *LC\_MESSAGES* category in the current locale.819 The implementation shall behave as if no function defined in this standard calls *strsignal()*.820 Since no return value is reserved to indicate an error, an application wishing to check for error  
821 situations should set *errno* to 0, then call *strsignal()*, then check *errno*.822 The *strsignal()* function need not be reentrant. A function that is not required to be reentrant is  
823 not required to be thread-safe.824 **RETURN VALUE**825 Upon successful completion, *strsignal()* shall return a pointer to a string. Otherwise if *signum* is  
826 not a valid signal number, the *strsignal()* function shall return a pointer to a string containing an  
827 unspecified message denoting an unknown signal.828 **ERRORS**

829 No errors are defined.

830 **EXAMPLES**

831 None.

832 **APPLICATION USAGE**833 The *strsignal()* function is part of the Extended Interfaces option and need not be available on all  
834 implementations.835 **RATIONALE**

836 None.

837 **FUTURE DIRECTIONS**

838 None.

839 **SEE ALSO**840 *perror()*, *psignal()*, the Base Definitions volume of IEEE Std 1003.1-2001, **<string.h>**841 **CHANGE HISTORY**

842 First released in Issue X



843 **NAME**844 `wcpcpy` — copy a wide character string, returning a pointer to its end845 **SYNOPSIS**846 

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

847 

```
wchar_t *wcpcpy(wchar_t *restrict dst, const wchar_t *restrict src);
```

848

849 **DESCRIPTION**850 The `wcpcpy()` function is the wide character equivalent of the `strcpy()` function. It shall copy the  
851 wide character string pointed to by `src`, including the terminating null wide-character code, into  
852 the array pointed to by `dst`.853 The application shall ensure that there is room for at least `wcslen(src)+1` wide characters in the  
854 `dst` array, and that the `src` and `dst` arrays do not overlap.855 **RETURN VALUE**856 The `wcpcpy()` function shall return a pointer to the last wide character written into the `dst` array,  
857 that is a pointer to the terminating null wide-character code. No return value is reserved to  
858 indicate an error.859 **ERRORS**

860 No errors are defined.

861 **EXAMPLES**

862 None.

863 **APPLICATION USAGE**864 The `wcpcpy()` function is part of the Extended Interfaces option and need not be available on all  
865 implementations.866 **RATIONALE**

867 None.

868 **FUTURE DIRECTIONS**

869 None.

870 **SEE ALSO**871 `strcpy()`, `wscopy()`, the Base Definitions volume of IEEE Std 1003.1-2001, `<wchar.h>`872 **CHANGE HISTORY**

873 **NAME**

874 wcpncpy — copy a fixed-size wide character string, returning a pointer to its end

875 **SYNOPSIS**876 

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

877 

```
wchar_t *wcpncpy(wchar_t restrict* dst, const wchar_t *restrict src,  
878 size_t n);
```

879

880 **DESCRIPTION**881 **Notes to Reviewers**882 *This section with side shading will not appear in the final copy. - Ed.*883 **Commentary on this function:**884 This page needs further work to improve the language to match the standard and also to tidy up  
885 some points (the current description makes it impossible to implement this function if  $n == 0$ ).886 The *wcpncpy()* function is the wide character equivalent of the *stpncpy()* function. It shall copy  
887 at most  $n$  wide characters from the string pointed to by *src*, including the terminating null wide-  
888 character code, into the array pointed to by *dst*. Exactly  $n$  wide characters shall be written into  
889 *dst*. If the length of *src* is smaller than  $n$  remaining characters for *dst* are filled in using the  
890 terminating null wide-character code. If the *src* array length is greater than, or equal to  $n$  then  $n$   
891 characters from *src* shall be copied to *dst* with no terminating null wide-character code in the *dst*  
892 array.893 The application shall ensure that there is room for at least  $n$  wide characters in the *dst* array, and  
894 that the *src* and *dst* arrays do not overlap.895 **RETURN VALUE**896 The *wcpncpy()* function shall return a pointer to the first null wide character that was written  
897 into the *dst* array, whatever the relation between *size* and the length of *src*. No return values are  
898 reserved to indicate an error.899 **ERRORS**

900 No errors are defined.

901 **EXAMPLES**

902 None.

903 **APPLICATION USAGE**904 The *wcpncpy()* function is part of the Extended Interfaces option and need not be available on all  
905 implementations.906 **RATIONALE**

907 None.

908 **FUTURE DIRECTIONS**

909 None.

910 **SEE ALSO**911 *stpncpy()*, *wpcpy()*, *wcsncpy()*, the Base Definitions volume of IEEE Std 1003.1-2001, `<wchar.h>`

912 **CHANGE HISTORY**  
913 First released in Issue X

914 **NAME**

915 wscasecmp — compare two wide character strings, ignoring case

916 **SYNOPSIS**917 `#include <wchar.h>`918 `int wscasecmp(const wchar_t *st1, const wchar_t *st2);`

919

920 **DESCRIPTION**921 **Notes to Reviewers**922 *This section with side shading will not appear in the final copy. - Ed.*923 **Commentary on this function:**924 Some issues with this man page. The return value section doesn't match the description.  
925 Language referring to st1 and st2, needs to be clear that its the the first wide character pointed to  
926 by st1 and st2927 The *wscasecmp()* function is the wide character equivalent of the *strcasecmp()* function. It shall  
928 compare the wide character string in *st1* with that found in *st2*. This comparison shall ignore  
929 case differences between the two strings.930 **RETURN VALUE**931 The *wscasecmp()* function shall return an integer containing the value 0 when the two strings  
932 are equal (ignoring case differences). The returned integer shall be positive when *st1* is greater  
933 than *st2*, ignoring case. The returned integer shall be negative when *st1* is smaller than *st2*,  
934 ignoring case. No return value is reserved to indicate an error.935 **ERRORS**

936 No errors are defined.

937 **EXAMPLES**

938 None.

939 **APPLICATION USAGE**940 The *wscasecmp()* function is part of the Extended Interfaces option and need not be available on  
941 all implementations.942 **RATIONALE**

943 None.

944 **FUTURE DIRECTIONS**

945 None.

946 **SEE ALSO**947 *strcasecmp()*, *wscmp()*, *wcsncasecmp()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
948 `<wchar.h>`949 **CHANGE HISTORY**

950 First released in Issue X

951 **NAME**

952       wcsdup — duplicate a wide character string

953 **SYNOPSIS**

954       #include &lt;wchar.h&gt;

955       wchar\_t \*wcsdup(const wchar\_t \*string);

956

957 **DESCRIPTION**958 **Notes to Reviewers**959       *This section with side shading will not appear in the final copy. - Ed.*960       **Commentary on this function:**961       Some issues with this man page. The description and return value sections do not actually state  
962       that the wide characters in the string argument are actually copied into the memory pointed to  
963       by the return value.964       The *wcsdup()* function is the wide character equivalent of the *strdup()* function. It shall allocate  
965       memory for a wide character string duplicate of that passed in *string*.966       The memory is allocated using *malloc()*, and should be freed using *free()*.967 **RETURN VALUE**968       Upon successful completion the *wcsdup()* function shall return a pointer to the newly allocated  
969       wide character string. Otherwise it shall return a null pointer and set *errno* to indicate the error.970 **ERRORS**971       The *wcsdup()* function shall fail if:

972       [ENOMEM]       memory large enough for the duplicate string could not be allocated.

973 **EXAMPLES**

974       None.

975 **APPLICATION USAGE**976       The *wcsdup()* function is part of the Extended Interfaces option and need not be available on all  
977       implementations.978 **RATIONALE**

979       None.

980 **FUTURE DIRECTIONS**

981       None.

982 **SEE ALSO**983       *strdup()*, *wscopy()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>984 **CHANGE HISTORY**

985       First released in Issue X

986 **NAME**

987       wscnsecmp — compare two fixed-size wide character strings, ignoring case

988 **SYNOPSIS**

989       #include &lt;wchar.h&gt;

990       int wscnsecmp(const wchar\_t \*st1, const wchar\_t \*st2, size\_t n);

991

992 **DESCRIPTION**993 **Notes to Reviewers**994       *This section with side shading will not appear in the final copy. - Ed.*995       **Commentary on this function:**996       Some issues with this man page. The return value section doesn't match the description. Phrases  
997       such as "truncated st1" need to be "the wide character string pointed to by st1".998       The *wscnsecmp()* function is the wide character equivalent of the *strncasecmp()* function. It  
999       shall compare at most *n* wide characters in *st1* to their counterparts in *st2*, ignoring case  
1000       differences.1001 **RETURN VALUE**1002       The *wscnsecmp()* function shall return an integer containing the value 0 when at most *n* wide  
1003       characters compare equal, ignoring case. This integer shall be a positive value when the  
1004       truncated *st1* is greater than the truncated *st2*, ignoring case. It shall be a negative value when  
1005       the truncated *st1* is less than the truncated *st2*, ignoring case. No return value is reserved to  
1006       indicate an error.1007 **ERRORS**

1008       No errors are defined.

1009 **EXAMPLES**

1010       None.

1011 **APPLICATION USAGE**1012       The *wscnsecmp()* function is part of the Extended Interfaces option and need not be available  
1013       on all implementations.1014 **RATIONALE**

1015       None.

1016 **FUTURE DIRECTIONS**

1017       None.

1018 **SEE ALSO**1019       *strncasecmp()*, *wscasecmp()*, *wscnsecmp()*, the Base Definitions volume of IEEE Std 1003.1-2001,  
1020       <wchar.h>1021 **CHANGE HISTORY**

1022       First released in Issue X

1023 **NAME**

1024       wcsnlen — determine the length of a fixed-sized wide character string

1025 **SYNOPSIS**

1026       #include &lt;wchar.h&gt;

1027       size\_t wcsnlen(const wchar\_t \*wcs, size\_t maxlen);

1028

1029 **DESCRIPTION**1030 **Notes to Reviewers**1031       *This section with side shading will not appear in the final copy. - Ed.*1032       **Commentary on this function:**1033       Some issues with this man page. The description and return value sections use non-standard  
1034       terms ("termination character", "size" of a wide character string). Uses of phrases such as "size of"  
1035       need to be updated to "length of a string"1036       The *wcsnlen()* function is the wide character equivalent of the *strlen()* function. It shall scan the  
1037       wide character string pointed to by the *wcs* argument up to at most *maxlen* wide characters,  
1038       looking for a termination character.1039 **RETURN VALUE**1040       The *wcsnlen()* function shall return an integer containing the smaller of either the size of the  
1041       wide character string pointed to by *wcs* or *maxlen*. No return value is reserved to indicate an  
1042       error.1043 **ERRORS**

1044       No errors are defined.

1045 **EXAMPLES**

1046       None.

1047 **APPLICATION USAGE**1048       The *wcsnlen()* function is part of the Extended Interfaces option and need not be available on all  
1049       implementations.1050 **RATIONALE**

1051       None.

1052 **FUTURE DIRECTIONS**

1053       None.

1054 **SEE ALSO**1055       *strlen()*, *wcslen()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>1056 **CHANGE HISTORY**

1057       First released in Issue X

1058 **NAME**

1059       wcsnrtoombs — convert wide-character string to multi-byte string

1060 **SYNOPSIS**

1061       #include &lt;wchar.h&gt;

1062       size\_t wcsnrtoombs(char \*dst, const wchar\_t \*\*src, size\_t nwc,  
1063                   size\_t len, mbstate\_t \*ps);  
10641065 **DESCRIPTION**1066 **Notes to Reviewers**1067       *This section with side shading will not appear in the final copy. - Ed.*1068       **Commentary on this function:**1069       The man page for this interface is incomplete and the references to wcsrtombs() are not  
1070       sufficient to understand how this is supposed to work in the general case. Need a much better  
1071       description1072       The *wcsnrtoombs()* function shall be equivalent to the *wcsrtombs()* function, except that the  
1073       conversion is limited to the first *nwc* wide characters.1074 **RETURN VALUE**1075       Refer to *wcsrtombs()*1076 **ERRORS**1077       Refer to *wcsrtombs()*1078 **EXAMPLES**

1079       None.

1080 **APPLICATION USAGE**1081       The *wcsnrtoombs()* function is part of the Extended Interfaces option and need not be available on  
1082       all implementations.1083 **RATIONALE**

1084       None.

1085 **FUTURE DIRECTIONS**

1086       None.

1087 **SEE ALSO**1088       *wcsrtombs()*, the Base Definitions volume of IEEE Std 1003.1-2001, <wchar.h>1089 **CHANGE HISTORY**

1090       First released in Issue X