[This page intentionally left blank]
## Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Introduction</th>
<th>General Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Chapter 2</td>
</tr>
<tr>
<td>1</td>
<td>Overview ..........................................................</td>
<td>Use and Implementation of Interfaces .........................................</td>
</tr>
<tr>
<td>1.1</td>
<td>Conformance ..................................................................</td>
<td>C Language in an Issue 4 Environment ...........................................</td>
</tr>
<tr>
<td>1.2</td>
<td>BASE Conformance .....................................................</td>
<td>Use of File System Interfaces ..................................................</td>
</tr>
<tr>
<td>1.2.1</td>
<td>X/Open UNIX Conformance ...........................................</td>
<td>The Compilation Environment .......................................................</td>
</tr>
<tr>
<td>1.2.2</td>
<td>Feature Groups ................................................................</td>
<td>X/Open UNIX Extensions ...........................................................</td>
</tr>
<tr>
<td>1.3</td>
<td>POSIX2 C-language Binding ...........................................</td>
<td>The X/Open Name Space .............................................................</td>
</tr>
<tr>
<td>1.3.1</td>
<td>Shared Memory ..........................................................</td>
<td>Error Numbers ............................................................................</td>
</tr>
<tr>
<td>1.3.2</td>
<td>Encryption ....................................................................</td>
<td>Additional Error Numbers ...........................................................</td>
</tr>
<tr>
<td>1.3.3</td>
<td>Enhanced Internationalisation ......................................</td>
<td>Standard I/O Streams .....................................................................</td>
</tr>
<tr>
<td>1.3.4</td>
<td>X/Open UNIX Extension ................................................</td>
<td>Interaction of File Descriptors and Standard I/O Streams ...............</td>
</tr>
<tr>
<td>1.3.5</td>
<td>Changes from Issue 3 ...................................................</td>
<td>STREAMS .........................................................................................</td>
</tr>
<tr>
<td>1.4</td>
<td>Changes from Issue 3 to Issue 4 ....................................</td>
<td>Accessing STREAMS .........................................................................</td>
</tr>
<tr>
<td>1.4.1</td>
<td>Changes from Issue 4 to Issue 4, Version 2 .................</td>
<td>Interprocess Communication ..........................................................</td>
</tr>
<tr>
<td>1.4.2</td>
<td>New Features ...................................................................</td>
<td>IPC General Description ...............................................................</td>
</tr>
<tr>
<td>1.4.3</td>
<td>To Be Withdrawn ...........................................................</td>
<td>Data Types .......................................................................................</td>
</tr>
<tr>
<td>1.4.4</td>
<td>Withdrawn ........................................................................</td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>Terminology .....................................................................</td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td>Relationship to Formal Standards ..................................</td>
<td></td>
</tr>
<tr>
<td>1.6.1</td>
<td>Relationship to Emerging Formal Standards ...................</td>
<td></td>
</tr>
<tr>
<td>1.6.2</td>
<td>Portability .......................................................................</td>
<td></td>
</tr>
<tr>
<td>1.7</td>
<td>Codes ...............................................................................</td>
<td></td>
</tr>
<tr>
<td>1.8</td>
<td>Format of Entries ..........................................................</td>
<td></td>
</tr>
</tbody>
</table>

System Interfaces and Headers Issue 4, Version 2
## Chapter 3 System Interfaces

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>alloc()</td>
<td>71</td>
</tr>
<tr>
<td>atexit()</td>
<td>58</td>
</tr>
<tr>
<td>atof()</td>
<td>59</td>
</tr>
<tr>
<td>atof()</td>
<td>60</td>
</tr>
<tr>
<td>atol()</td>
<td>61</td>
</tr>
<tr>
<td>basename()</td>
<td>62</td>
</tr>
<tr>
<td>bcopy()</td>
<td>63</td>
</tr>
<tr>
<td>bsearch()</td>
<td>64</td>
</tr>
<tr>
<td>bsd_signal()</td>
<td>66</td>
</tr>
<tr>
<td>bsearch()</td>
<td>67</td>
</tr>
<tr>
<td>bzero()</td>
<td>70</td>
</tr>
<tr>
<td>ceil()</td>
<td>71</td>
</tr>
<tr>
<td>catclose()</td>
<td>72</td>
</tr>
<tr>
<td>catopen()</td>
<td>74</td>
</tr>
<tr>
<td>cbrt()</td>
<td>76</td>
</tr>
<tr>
<td>cfgetispeed()</td>
<td>77</td>
</tr>
<tr>
<td>cfgetospeed()</td>
<td>78</td>
</tr>
<tr>
<td>cfsetispeed()</td>
<td>79</td>
</tr>
<tr>
<td>cfsetospeed()</td>
<td>80</td>
</tr>
<tr>
<td>chdir()</td>
<td>81</td>
</tr>
<tr>
<td>chmod()</td>
<td>82</td>
</tr>
<tr>
<td>chown()</td>
<td>84</td>
</tr>
<tr>
<td>chroot()</td>
<td>86</td>
</tr>
<tr>
<td>clearerr()</td>
<td>88</td>
</tr>
<tr>
<td>clock()</td>
<td>90</td>
</tr>
<tr>
<td>close()</td>
<td>91</td>
</tr>
<tr>
<td>closedir()</td>
<td>92</td>
</tr>
<tr>
<td>closelog()</td>
<td>94</td>
</tr>
<tr>
<td>compile()</td>
<td>95</td>
</tr>
<tr>
<td>confstr()</td>
<td>98</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
</tr>
<tr>
<td>getopt()</td>
<td>251</td>
</tr>
<tr>
<td>getpagesize()</td>
<td>254</td>
</tr>
<tr>
<td>getpass()</td>
<td>255</td>
</tr>
<tr>
<td>getpgid()</td>
<td>257</td>
</tr>
<tr>
<td>getpgrp()</td>
<td>258</td>
</tr>
<tr>
<td>getpid()</td>
<td>259</td>
</tr>
<tr>
<td>getpmsg()</td>
<td>260</td>
</tr>
<tr>
<td>getppid()</td>
<td>261</td>
</tr>
<tr>
<td>getpriority()</td>
<td>262</td>
</tr>
<tr>
<td>getpwent()</td>
<td>264</td>
</tr>
<tr>
<td>getpwnam()</td>
<td>265</td>
</tr>
<tr>
<td>getpwuid()</td>
<td>267</td>
</tr>
<tr>
<td>getrlimit()</td>
<td>269</td>
</tr>
<tr>
<td>getusage()</td>
<td>271</td>
</tr>
<tr>
<td>gets()</td>
<td>272</td>
</tr>
<tr>
<td>getsid()</td>
<td>273</td>
</tr>
<tr>
<td>getsubopt()</td>
<td>274</td>
</tr>
<tr>
<td>gettimeofday()</td>
<td>275</td>
</tr>
<tr>
<td>getuid()</td>
<td>276</td>
</tr>
<tr>
<td>getutxent()</td>
<td>277</td>
</tr>
<tr>
<td>getw()</td>
<td>278</td>
</tr>
<tr>
<td>getwc()</td>
<td>279</td>
</tr>
<tr>
<td>getwchar()</td>
<td>280</td>
</tr>
<tr>
<td>getwid()</td>
<td>281</td>
</tr>
<tr>
<td>glob()</td>
<td>282</td>
</tr>
<tr>
<td>gmtime()</td>
<td>286</td>
</tr>
<tr>
<td>grantpt()</td>
<td>287</td>
</tr>
<tr>
<td>hcreate()</td>
<td>288</td>
</tr>
<tr>
<td>hdestroy()</td>
<td>289</td>
</tr>
<tr>
<td>hsearch()</td>
<td>290</td>
</tr>
<tr>
<td>hypot()</td>
<td>293</td>
</tr>
<tr>
<td>iconv()</td>
<td>294</td>
</tr>
<tr>
<td>iconv_close()</td>
<td>296</td>
</tr>
<tr>
<td>iconv_open()</td>
<td>297</td>
</tr>
<tr>
<td>ilogb()</td>
<td>298</td>
</tr>
<tr>
<td>index()</td>
<td>299</td>
</tr>
<tr>
<td>initstate()</td>
<td>300</td>
</tr>
<tr>
<td>insqque()</td>
<td>302</td>
</tr>
<tr>
<td>ioctl()</td>
<td>304</td>
</tr>
<tr>
<td>isalnum()</td>
<td>315</td>
</tr>
<tr>
<td>isalpha()</td>
<td>316</td>
</tr>
<tr>
<td>isascii()</td>
<td>317</td>
</tr>
<tr>
<td>isastream()</td>
<td>318</td>
</tr>
<tr>
<td>isatty()</td>
<td>319</td>
</tr>
<tr>
<td>iscntrl()</td>
<td>320</td>
</tr>
<tr>
<td>isdigit()</td>
<td>321</td>
</tr>
<tr>
<td>isgraph()</td>
<td>322</td>
</tr>
<tr>
<td>islower()</td>
<td>323</td>
</tr>
</tbody>
</table>
isnan() ................................................................. 324
isprint() ............................................................. 325
ispunct() ............................................................. 326
isspace() ............................................................. 327
isupper() ............................................................. 328
iswalnum() .......................................................... 329
iswalpha() ........................................................... 330
iswcntrl() ............................................................ 331
iswctype() ........................................................... 332
iswdigit() ............................................................. 333
iswgraph() ........................................................... 334
iswlower() ............................................................ 335
iswprint() ............................................................ 336
iswpunct() ............................................................ 337
iswspace() ............................................................ 338
iswupper() ............................................................ 339
iswxdigit() ............................................................ 340
isxdigit() .............................................................. 341
j0() ........................................................................... 342
jrand48() .................................................................. 343
kill() ........................................................................ 344
killpg() .................................................................... 346
l64a() ....................................................................... 347
labs() .......................................................................... 348
lchown() ..................................................................... 349
lcong48() .................................................................... 350
ldexp() ......................................................................... 351
ldiv() ........................................................................... 352
lfind() ......................................................................... 353
lgamma() ...................................................................... 354
link() ............................................................................ 355
loc1 ............................................................................. 357
localeconv() ............................................................ 358
localtime() .............................................................. 359
lockf() .......................................................................... 360
locs .............................................................................. 362
log() ............................................................................ 364
log10() ........................................................................ 365
log1p() ......................................................................... 366
logb() ........................................................................... 367
_longjmp() .................................................................... 368
longjmp() ....................................................................... 369
lrand48() ...................................................................... 370
lsearch() ....................................................................... 371
lseek() .......................................................................... 372
lstat() ........................................................................... 373
makecontext() ............................................................ 374
malloc() ....................................................................... 375
makecontext() ............................................................ 376
makecontext() ............................................................ 377
malloc() ....................................................................... 378
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbedtls()</td>
<td>379</td>
</tr>
<tr>
<td>mbstowcs()</td>
<td>380</td>
</tr>
<tr>
<td>mbtowc()</td>
<td>381</td>
</tr>
<tr>
<td>memccpy()</td>
<td>382</td>
</tr>
<tr>
<td>memchr()</td>
<td>383</td>
</tr>
<tr>
<td>memcmp()</td>
<td>384</td>
</tr>
<tr>
<td>memcpy()</td>
<td>385</td>
</tr>
<tr>
<td>memmove()</td>
<td>386</td>
</tr>
<tr>
<td>memset()</td>
<td>387</td>
</tr>
<tr>
<td>mkdir()</td>
<td>388</td>
</tr>
<tr>
<td>mktime()</td>
<td>389</td>
</tr>
<tr>
<td>mknode()</td>
<td>390</td>
</tr>
<tr>
<td>mkod()</td>
<td>391</td>
</tr>
<tr>
<td>mkstemp()</td>
<td>392</td>
</tr>
<tr>
<td>mktemp()</td>
<td>393</td>
</tr>
<tr>
<td>mktime()</td>
<td>394</td>
</tr>
<tr>
<td>mmap()</td>
<td>395</td>
</tr>
<tr>
<td>modf()</td>
<td>396</td>
</tr>
<tr>
<td>mprotect()</td>
<td>397</td>
</tr>
<tr>
<td>mtime()</td>
<td>398</td>
</tr>
<tr>
<td>mprotect()</td>
<td>399</td>
</tr>
<tr>
<td>msgctl()</td>
<td>400</td>
</tr>
<tr>
<td>msgget()</td>
<td>401</td>
</tr>
<tr>
<td>msgget()</td>
<td>402</td>
</tr>
<tr>
<td>msgrcv()</td>
<td>403</td>
</tr>
<tr>
<td>msync()</td>
<td>404</td>
</tr>
<tr>
<td>munmap()</td>
<td>405</td>
</tr>
<tr>
<td>nextafter()</td>
<td>406</td>
</tr>
<tr>
<td>nftw()</td>
<td>407</td>
</tr>
<tr>
<td>nfnv()</td>
<td>408</td>
</tr>
<tr>
<td>nice()</td>
<td>409</td>
</tr>
<tr>
<td>nl_langinfo()</td>
<td>410</td>
</tr>
<tr>
<td>nrand48()</td>
<td>411</td>
</tr>
<tr>
<td>open()</td>
<td>412</td>
</tr>
<tr>
<td>opendir()</td>
<td>413</td>
</tr>
<tr>
<td>openlog()</td>
<td>414</td>
</tr>
<tr>
<td>optarg</td>
<td>415</td>
</tr>
<tr>
<td>pathconf()</td>
<td>416</td>
</tr>
<tr>
<td>pause()</td>
<td>417</td>
</tr>
<tr>
<td>pclose()</td>
<td>418</td>
</tr>
<tr>
<td>perror()</td>
<td>419</td>
</tr>
<tr>
<td>pipe()</td>
<td>420</td>
</tr>
<tr>
<td>poll()</td>
<td>421</td>
</tr>
<tr>
<td>popen()</td>
<td>422</td>
</tr>
<tr>
<td>pow()</td>
<td>423</td>
</tr>
<tr>
<td>printf()</td>
<td>424</td>
</tr>
<tr>
<td>printw()</td>
<td>425</td>
</tr>
<tr>
<td>putc()</td>
<td>426</td>
</tr>
<tr>
<td>putchar()</td>
<td>427</td>
</tr>
<tr>
<td>putenv()</td>
<td>428</td>
</tr>
<tr>
<td>putmsg()</td>
<td>429</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------</td>
<td>------</td>
</tr>
<tr>
<td>puts()</td>
<td>449</td>
</tr>
<tr>
<td>pututxline()</td>
<td>450</td>
</tr>
<tr>
<td>putw()</td>
<td>451</td>
</tr>
<tr>
<td>putwc()</td>
<td>452</td>
</tr>
<tr>
<td>putwchar()</td>
<td>453</td>
</tr>
<tr>
<td>qsort()</td>
<td>454</td>
</tr>
<tr>
<td>raise()</td>
<td>455</td>
</tr>
<tr>
<td>rand()</td>
<td>456</td>
</tr>
<tr>
<td>random()</td>
<td>458</td>
</tr>
<tr>
<td>read()</td>
<td>459</td>
</tr>
<tr>
<td>readdir()</td>
<td>463</td>
</tr>
<tr>
<td>readdir()</td>
<td>465</td>
</tr>
<tr>
<td>readv()</td>
<td>466</td>
</tr>
<tr>
<td>realloc()</td>
<td>467</td>
</tr>
<tr>
<td>realpath()</td>
<td>469</td>
</tr>
<tr>
<td>re_comp()</td>
<td>470</td>
</tr>
<tr>
<td>regcmp()</td>
<td>472</td>
</tr>
<tr>
<td>regcomp()</td>
<td>474</td>
</tr>
<tr>
<td>regex()</td>
<td>479</td>
</tr>
<tr>
<td>regexp()</td>
<td>480</td>
</tr>
<tr>
<td>remainder()</td>
<td>486</td>
</tr>
<tr>
<td>remove()</td>
<td>487</td>
</tr>
<tr>
<td>remque()</td>
<td>488</td>
</tr>
<tr>
<td>rename()</td>
<td>489</td>
</tr>
<tr>
<td>rewind()</td>
<td>492</td>
</tr>
<tr>
<td>rewinddir()</td>
<td>493</td>
</tr>
<tr>
<td>rindex()</td>
<td>494</td>
</tr>
<tr>
<td>rint()</td>
<td>495</td>
</tr>
<tr>
<td>rmdir()</td>
<td>496</td>
</tr>
<tr>
<td>sbrk()</td>
<td>498</td>
</tr>
<tr>
<td>scalb()</td>
<td>499</td>
</tr>
<tr>
<td>scanf()</td>
<td>500</td>
</tr>
<tr>
<td>seed48()</td>
<td>501</td>
</tr>
<tr>
<td>seekdir()</td>
<td>502</td>
</tr>
<tr>
<td>select()</td>
<td>503</td>
</tr>
<tr>
<td>semct1()</td>
<td>506</td>
</tr>
<tr>
<td>semget()</td>
<td>509</td>
</tr>
<tr>
<td>semop()</td>
<td>511</td>
</tr>
<tr>
<td>setbuf()</td>
<td>514</td>
</tr>
<tr>
<td>setcontext()</td>
<td>515</td>
</tr>
<tr>
<td>setgid()</td>
<td>516</td>
</tr>
<tr>
<td>setgrent()</td>
<td>517</td>
</tr>
<tr>
<td>setitimer()</td>
<td>518</td>
</tr>
<tr>
<td>_setjmp()</td>
<td>519</td>
</tr>
<tr>
<td>setjmp()</td>
<td>520</td>
</tr>
<tr>
<td>setkey()</td>
<td>522</td>
</tr>
<tr>
<td>setlocale()</td>
<td>523</td>
</tr>
<tr>
<td>setlogmask()</td>
<td>525</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
</tr>
<tr>
<td>setpgid()</td>
<td>526</td>
</tr>
<tr>
<td>setpgroup()</td>
<td>528</td>
</tr>
<tr>
<td>setpriority()</td>
<td>529</td>
</tr>
<tr>
<td>setregid()</td>
<td>530</td>
</tr>
<tr>
<td>setreuid()</td>
<td>531</td>
</tr>
<tr>
<td>setrtrimt()</td>
<td>532</td>
</tr>
<tr>
<td>setsid()</td>
<td>533</td>
</tr>
<tr>
<td>setstate()</td>
<td>534</td>
</tr>
<tr>
<td>setuid()</td>
<td>535</td>
</tr>
<tr>
<td>setutsxent()</td>
<td>536</td>
</tr>
<tr>
<td>setvbuf()</td>
<td>537</td>
</tr>
<tr>
<td>shmat()</td>
<td>538</td>
</tr>
<tr>
<td>shmctl()</td>
<td>540</td>
</tr>
<tr>
<td>shmct1()</td>
<td>542</td>
</tr>
<tr>
<td>shmdt()</td>
<td>543</td>
</tr>
<tr>
<td>shmget()</td>
<td>544</td>
</tr>
<tr>
<td>sigaction()</td>
<td>545</td>
</tr>
<tr>
<td>sigaddset()</td>
<td>553</td>
</tr>
<tr>
<td>sigaltsf()</td>
<td>554</td>
</tr>
<tr>
<td>sigdelset()</td>
<td>556</td>
</tr>
<tr>
<td>sigemptyset()</td>
<td>557</td>
</tr>
<tr>
<td>sigfillset()</td>
<td>558</td>
</tr>
<tr>
<td>sighold()</td>
<td>559</td>
</tr>
<tr>
<td>siginterrupt()</td>
<td>560</td>
</tr>
<tr>
<td>sigismember()</td>
<td>561</td>
</tr>
<tr>
<td>siglongjmp()</td>
<td>562</td>
</tr>
<tr>
<td>signal()</td>
<td>563</td>
</tr>
<tr>
<td>sign()</td>
<td>566</td>
</tr>
<tr>
<td>signgam()</td>
<td>567</td>
</tr>
<tr>
<td>sigpause()</td>
<td>568</td>
</tr>
<tr>
<td>sigpending()</td>
<td>569</td>
</tr>
<tr>
<td>sigprocmask()</td>
<td>570</td>
</tr>
<tr>
<td>sigrelse()</td>
<td>571</td>
</tr>
<tr>
<td>sigsetjmp()</td>
<td>572</td>
</tr>
<tr>
<td>sigstack()</td>
<td>574</td>
</tr>
<tr>
<td>sigsuspend()</td>
<td>576</td>
</tr>
<tr>
<td>sin()</td>
<td>577</td>
</tr>
<tr>
<td>sinh()</td>
<td>578</td>
</tr>
<tr>
<td>sleep()</td>
<td>579</td>
</tr>
<tr>
<td>sprintf()</td>
<td>581</td>
</tr>
<tr>
<td>sqrt()</td>
<td>582</td>
</tr>
<tr>
<td>srand()</td>
<td>583</td>
</tr>
<tr>
<td>srand48()</td>
<td>584</td>
</tr>
<tr>
<td>random()</td>
<td>585</td>
</tr>
<tr>
<td>sscanf()</td>
<td>586</td>
</tr>
<tr>
<td>stat()</td>
<td>587</td>
</tr>
<tr>
<td>statvfs()</td>
<td>588</td>
</tr>
<tr>
<td>stdin()</td>
<td>589</td>
</tr>
<tr>
<td>step()</td>
<td>590</td>
</tr>
<tr>
<td>strcasecmp()</td>
<td>591</td>
</tr>
<tr>
<td>strncasecmp()</td>
<td>592</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
</tr>
<tr>
<td>strcat()</td>
<td>593</td>
</tr>
<tr>
<td>strlen()</td>
<td>594</td>
</tr>
<tr>
<td>strcmp()</td>
<td>595</td>
</tr>
<tr>
<td>strcoll()</td>
<td>596</td>
</tr>
<tr>
<td>strcpy()</td>
<td>597</td>
</tr>
<tr>
<td>strcasecmp()</td>
<td>598</td>
</tr>
<tr>
<td>strdup()</td>
<td>599</td>
</tr>
<tr>
<td>strerror()</td>
<td>600</td>
</tr>
<tr>
<td>strfmon()</td>
<td>601</td>
</tr>
<tr>
<td>strftime()</td>
<td>605</td>
</tr>
<tr>
<td>strlen()</td>
<td>608</td>
</tr>
<tr>
<td>strncasecmp()</td>
<td>609</td>
</tr>
<tr>
<td>strncat()</td>
<td>610</td>
</tr>
<tr>
<td>strncmp()</td>
<td>611</td>
</tr>
<tr>
<td>strncpy()</td>
<td>612</td>
</tr>
<tr>
<td>strpbrk()</td>
<td>613</td>
</tr>
<tr>
<td>strstr()</td>
<td>614</td>
</tr>
<tr>
<td>strrchr()</td>
<td>617</td>
</tr>
<tr>
<td>strspn()</td>
<td>618</td>
</tr>
<tr>
<td>strrstr()</td>
<td>619</td>
</tr>
<tr>
<td>strtol()</td>
<td>620</td>
</tr>
<tr>
<td>strtok()</td>
<td>622</td>
</tr>
<tr>
<td>strtol()</td>
<td>623</td>
</tr>
<tr>
<td>strtof()</td>
<td>625</td>
</tr>
<tr>
<td>strxfrm()</td>
<td>627</td>
</tr>
<tr>
<td>swab()</td>
<td>629</td>
</tr>
<tr>
<td>swapcontext()</td>
<td>630</td>
</tr>
<tr>
<td>symlink()</td>
<td>631</td>
</tr>
<tr>
<td>sync()</td>
<td>633</td>
</tr>
<tr>
<td>sysconf()</td>
<td>634</td>
</tr>
<tr>
<td>syslog()</td>
<td>637</td>
</tr>
<tr>
<td>system()</td>
<td>638</td>
</tr>
<tr>
<td>tan()</td>
<td>640</td>
</tr>
<tr>
<td>tanh()</td>
<td>641</td>
</tr>
<tr>
<td>tcdrain()</td>
<td>642</td>
</tr>
<tr>
<td>tcflow()</td>
<td>644</td>
</tr>
<tr>
<td>tcflush()</td>
<td>646</td>
</tr>
<tr>
<td>tcgetattr()</td>
<td>648</td>
</tr>
<tr>
<td>tcgetpgrp()</td>
<td>649</td>
</tr>
<tr>
<td>tcgetsid()</td>
<td>650</td>
</tr>
<tr>
<td>tcsendbreak()</td>
<td>651</td>
</tr>
<tr>
<td>tcsetattr()</td>
<td>653</td>
</tr>
<tr>
<td>tcsetpgrp()</td>
<td>655</td>
</tr>
<tr>
<td>tdelete()</td>
<td>656</td>
</tr>
<tr>
<td>telldir()</td>
<td>657</td>
</tr>
<tr>
<td>tempnam()</td>
<td>658</td>
</tr>
<tr>
<td>tfind()</td>
<td>659</td>
</tr>
<tr>
<td>time()</td>
<td>660</td>
</tr>
</tbody>
</table>
Contents

times() .............................................................................................................. 661
timezone ........................................................................................................ 662
tmpfile() ....................................................................................................... 663
tmpnam() ....................................................................................................... 664
toascii() ......................................................................................................... 665
_totolower() .................................................................................................... 666
tolower() ......................................................................................................... 667
_totoupper() .................................................................................................... 668
toupper() .......................................................................................................... 669
towlower() ....................................................................................................... 670
towupper() ....................................................................................................... 671
truncate() ......................................................................................................... 672
tsearch() .......................................................................................................... 673
ttyname() ......................................................................................................... 677
tyslot() ............................................................................................................. 678
twalk() .............................................................................................................. 679
tzname ............................................................................................................. 680
tzset() .............................................................................................................. 681
ualarm() ............................................................................................................ 683
ulimit() ............................................................................................................. 684
umask() ............................................................................................................ 685
uname() ............................................................................................................. 686
ungetc() ............................................................................................................ 687
ungetwc() ......................................................................................................... 688
unlink() ............................................................................................................ 689
unlockpt() ......................................................................................................... 691
usleep() ............................................................................................................ 692
utime() ............................................................................................................. 693
utimes() .............................................................................................................. 695
valloc() ............................................................................................................ 696
vfork() .............................................................................................................. 697
vfprintf() ......................................................................................................... 699
wait() ................................................................................................................. 700
wait3() .............................................................................................................. 704
waitid() ............................................................................................................. 705
waitpid() .......................................................................................................... 707
wscat() ............................................................................................................. 708
wesch() .............................................................................................................. 709
wescmp() ......................................................................................................... 710
wescoll() .......................................................................................................... 711
wescpy() ............................................................................................................ 712
wescsprint() ..................................................................................................... 713
wescftime() ....................................................................................................... 714
weslen() ............................................................................................................ 715
wesncat() ......................................................................................................... 716
wesncmp() .......................................................................................................... 717
wesncpy() .......................................................................................................... 718
wespbrk() .......................................................................................................... 719
<table>
<thead>
<tr>
<th>Header</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>assert.h</td>
<td>746</td>
</tr>
<tr>
<td>assert.h</td>
<td>747</td>
</tr>
<tr>
<td>cpio.h</td>
<td>748</td>
</tr>
<tr>
<td>cpio.h</td>
<td>749</td>
</tr>
<tr>
<td>dirent.h</td>
<td>750</td>
</tr>
<tr>
<td>errno.h</td>
<td>751</td>
</tr>
<tr>
<td>errno.h</td>
<td>752</td>
</tr>
<tr>
<td>float.h</td>
<td>753</td>
</tr>
<tr>
<td>fmtmsg.h</td>
<td>754</td>
</tr>
<tr>
<td>fnmatch.h</td>
<td>755</td>
</tr>
<tr>
<td>ftw.h</td>
<td>756</td>
</tr>
<tr>
<td>glob.h</td>
<td>757</td>
</tr>
<tr>
<td>grp.h</td>
<td>758</td>
</tr>
<tr>
<td>iconv.h</td>
<td>759</td>
</tr>
<tr>
<td>langinfo.h</td>
<td>760</td>
</tr>
<tr>
<td>libgen.h</td>
<td>761</td>
</tr>
<tr>
<td>limits.h</td>
<td>762</td>
</tr>
<tr>
<td>locale.h</td>
<td>763</td>
</tr>
<tr>
<td>math.h</td>
<td>764</td>
</tr>
<tr>
<td>monetary.h</td>
<td>765</td>
</tr>
<tr>
<td>ndbm.h</td>
<td>766</td>
</tr>
<tr>
<td>nl_types.h</td>
<td>767</td>
</tr>
<tr>
<td>poll.h</td>
<td>768</td>
</tr>
<tr>
<td>pwd.h</td>
<td>769</td>
</tr>
<tr>
<td>regex.h</td>
<td>770</td>
</tr>
<tr>
<td>re_comp.h</td>
<td>771</td>
</tr>
<tr>
<td>reexp.h</td>
<td>772</td>
</tr>
<tr>
<td>search.h</td>
<td>773</td>
</tr>
<tr>
<td>setjmp.h</td>
<td>774</td>
</tr>
<tr>
<td>signal.h</td>
<td>775</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>776</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>777</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>778</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>779</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>780</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>781</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>782</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>783</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>784</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>785</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>786</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>787</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>788</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>789</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>790</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>791</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>792</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>793</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>794</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>795</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>796</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>797</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>798</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>799</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>800</td>
</tr>
<tr>
<td>stdarg.h</td>
<td>801</td>
</tr>
<tr>
<td>Header</td>
<td>Page</td>
</tr>
<tr>
<td>---------------</td>
<td>------</td>
</tr>
<tr>
<td><code>&lt;stddef.h&gt;</code></td>
<td>803</td>
</tr>
<tr>
<td><code>&lt;stdio.h&gt;</code></td>
<td>804</td>
</tr>
<tr>
<td><code>&lt;stdlib.h&gt;</code></td>
<td>807</td>
</tr>
<tr>
<td><code>&lt;string.h&gt;</code></td>
<td>810</td>
</tr>
<tr>
<td><code>&lt;strings.h&gt;</code></td>
<td>812</td>
</tr>
<tr>
<td><code>&lt;stropts.h&gt;</code></td>
<td>813</td>
</tr>
<tr>
<td><code>&lt;syslog.h&gt;</code></td>
<td>818</td>
</tr>
<tr>
<td><code>&lt;sys/ipc.h&gt;</code></td>
<td>820</td>
</tr>
<tr>
<td><code>&lt;sys/mman.h&gt;</code></td>
<td>822</td>
</tr>
<tr>
<td><code>&lt;sys/msg.h&gt;</code></td>
<td>823</td>
</tr>
<tr>
<td><code>&lt;sys/resource.h&gt;</code></td>
<td>825</td>
</tr>
<tr>
<td><code>&lt;sys勇敢.h&gt;</code></td>
<td>827</td>
</tr>
<tr>
<td><code>&lt;sys/shm.h&gt;</code></td>
<td>829</td>
</tr>
<tr>
<td><code>&lt;sys/stat.h&gt;</code></td>
<td>830</td>
</tr>
<tr>
<td><code>&lt;sys/statvfs.h&gt;</code></td>
<td>833</td>
</tr>
<tr>
<td><code>&lt;sys/time.h&gt;</code></td>
<td>834</td>
</tr>
<tr>
<td><code>&lt;sys/timeb.h&gt;</code></td>
<td>836</td>
</tr>
<tr>
<td><code>&lt;sys/times.h&gt;</code></td>
<td>837</td>
</tr>
<tr>
<td><code>&lt;sys/types.h&gt;</code></td>
<td>838</td>
</tr>
<tr>
<td><code>&lt;sys/uio.h&gt;</code></td>
<td>840</td>
</tr>
<tr>
<td><code>&lt;sys/utsname.h&gt;</code></td>
<td>841</td>
</tr>
<tr>
<td><code>&lt;sys/wait.h&gt;</code></td>
<td>842</td>
</tr>
<tr>
<td><code>&lt;tar.h&gt;</code></td>
<td>844</td>
</tr>
<tr>
<td><code>&lt;termios.h&gt;</code></td>
<td>846</td>
</tr>
<tr>
<td><code>&lt;time.h&gt;</code></td>
<td>851</td>
</tr>
<tr>
<td><code>&lt;ucontext.h&gt;</code></td>
<td>853</td>
</tr>
<tr>
<td><code>&lt;ulimit.h&gt;</code></td>
<td>854</td>
</tr>
<tr>
<td><code>&lt;unistd.h&gt;</code></td>
<td>863</td>
</tr>
<tr>
<td><code>&lt;utmpx.h&gt;</code></td>
<td>864</td>
</tr>
<tr>
<td><code>&lt;varargs.h&gt;</code></td>
<td>865</td>
</tr>
<tr>
<td><code>&lt;wchar.h&gt;</code></td>
<td>867</td>
</tr>
<tr>
<td><code>&lt;wordexp.h&gt;</code></td>
<td>869</td>
</tr>
</tbody>
</table>

Index ........................................................................................................ 871
X/Open

X/Open is an independent, worldwide, open systems organisation supported by most of the world’s largest information systems suppliers, user organisations and software companies. Its mission is to bring to users greater value from computing, through the practical implementation of open systems.

X/Open’s strategy for achieving this goal is to combine existing and emerging standards into a comprehensive, integrated, high-value and usable open system environment, called the Common Applications Environment (CAE). This environment covers the standards, above the hardware level, that are needed to support open systems. It provides for portability and interoperability of applications, and so protects investment in existing software while enabling additions and enhancements. It also allows users to move between systems with a minimum of retraining.

X/Open defines this CAE in a set of specifications which include an evolving portfolio of application programming interfaces (APIs) which significantly enhance portability of application programs at the source code level, along with definitions of and references to protocols and protocol profiles which significantly enhance the interoperability of applications and systems.

The X/Open CAE is implemented in real products and recognised by a distinctive trade mark — the X/Open brand — that is licensed by X/Open and may be used on products which have demonstrated their conformance.

X/Open Technical Publications

X/Open publishes a wide range of technical literature, the main part of which is focussed on specification development, but which also includes Guides, Snapshots, Technical Studies,Branding/Testing documents, industry surveys, and business titles.

There are two types of X/Open specification:

- **CAE Specifications**

  CAE (Common Applications Environment) specifications are the stable specifications that form the basis for X/Open-branded products. These specifications are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement an X/Open CAE specification can enjoy the benefits of a single, widely supported standard. In addition, they can demonstrate compliance with the majority of X/Open CAE specifications once these specifications are referenced in an X/Open component or profile definition and included in the X/Open branding programme.

  CAE specifications are published as soon as they are developed, not published to coincide with the launch of a particular X/Open brand. By making its specifications available in this way, X/Open makes it possible for conformance products to be developed as soon as is practicable, so enhancing the value of the X/Open brand as a procurement aid to users.
Preliminary Specifications

These specifications, which often address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations, are released in a controlled manner for the purpose of validation through implementation of products. A Preliminary specification is not a draft specification. In fact, it is as stable as X/Open can make it, and on publication has gone through the same rigorous X/Open development and review procedures as a CAE specification.

Preliminary specifications are analogous to the trial-use standards issued by formal standards organisations, and product development teams are encouraged to develop products on the basis of them. However, because of the nature of the technology that a Preliminary specification is addressing, it may be untried in multiple independent implementations, and may therefore change before being published as a CAE specification. There is always the intent to progress to a corresponding CAE specification, but the ability to do so depends on consensus among X/Open members. In all cases, any resulting CAE specification is made as upwards-compatible as possible. However, complete upwards-compatibility from the Preliminary to the CAE specification cannot be guaranteed.

In addition, X/Open publishes:

• Guides

These provide information that X/Open believes is useful in the evaluation, procurement, development or management of open systems, particularly those that are X/Open-compliant. X/Open Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming X/Open conformance.

• Technical Studies

X/Open Technical Studies present results of analyses performed by X/Open on subjects of interest in areas relevant to X/Open's Technical Programme. They are intended to communicate the findings to the outside world and, where appropriate, stimulate discussion and actions by other bodies and the industry in general.

• Snapshots

These provide a mechanism for X/Open to disseminate information on its current direction and thinking, in advance of possible development of a Specification, Guide or Technical Study. The intention is to stimulate industry debate and prototyping, and solicit feedback. A Snapshot represents the interim results of an X/Open technical activity. Although at the time of its publication, there may be an intention to progress the activity towards publication of a Specification, Guide or Technical Study, X/Open is a consensus organisation, and makes no commitment regarding future development and further publication. Similarly, a Snapshot does not represent any commitment by X/Open members to develop any specific products.

Versions and Issues of Specifications

As with all live documents, CAE Specifications require revision, in this case as the subject technology develops and to align with emerging associated international standards. X/Open makes a distinction between revised specifications which are fully backward compatible and those which are not:

• a new Version indicates that this publication includes all the same (unchanged) definitive information from the previous publication of that title, but also includes extensions or additional information. As such, it replaces the previous publication.
• a new Issue does include changes to the definitive information contained in the previous publication of that title (and may also include extensions or additional information). As such, X/Open maintains both the previous and new issue as current publications.

Corrigenda

Most X/Open publications deal with technology at the leading edge of open systems development. Feedback from implementation experience gained from using these publications occasionally uncovers errors or inconsistencies. Significant errors or recommended solutions to reported problems are communicated by means of Corrigenda.

The reader of this document is advised to check periodically if any Corrigenda apply to this publication. This may be done either by email to the X/Open info-server or by checking the Corrigenda list in the latest X/Open Publications Price List.

To request Corrigenda information by email, send a message to info-server@xopen.co.uk with the following in the Subject line:

request corrigenda; topic index

This will return the index of publications for which Corrigenda exist.

This Document

This specification is one of a set of X/Open CAE Specifications (see above) defining the X/Open System Interface (XSI) Operating System requirements:

• System Interface Definitions, Issue 4, Version 2 (the XBD specification)
• Commands and Utilities, Issue 4, Version 2 (the XCU specification)
• System Interfaces and Headers, Issue 4, Version 2 (this document).

This document describes the interfaces offered to application programs by XSI-conformant systems. Readers are expected to be experienced C language programmers, and to be familiar with the XBD specification. This specification is structured as follows:

• Chapter 1 explains the status of the document and its relationship to formal standards.
• Chapter 2 contains important notes, terms and caveats relating to the rest of the document.
• Chapter 3 defines the functional interfaces to the XSI-conformant system.
• Chapter 4 defines the contents of headers which declare constants, macros and data structures that are needed by programs using the services provided by Chapter 3.

Comprehensive references are available in the index.

Typographical Conventions

The following typographical conventions are used throughout this document:

• **Bold** font is used in text for options to commands, filenames, keywords, type names, data structures and their members.

• *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  — command operands, command option-arguments or variable names, for example, substitutable argument prototypes
  — environment variables, which are also shown in capitals
— utility names
— external variables, such as \textit{errno}
— functions; these are shown as follows: \textit{name()}; names without parentheses are C external variables, C function family names, utility names, command operands or command option-arguments.

- Normal font is used for the names of constants and literals.
- The notation \texttt{<file.h>} indicates a header file.
- Names surrounded by braces, for example, \{ARG\_MAX\}, represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C \texttt{#define} construct.
- The notation \texttt{[EABCD]} is used to identify an error value EABCD.
- Syntax, code examples and user input in interactive examples are shown in \textbf{fixed width} font. Brackets shown in this font, [], are part of the syntax and do not indicate optional items. In syntax the | symbol is used to separate alternatives, and ellipses (\ldots) are used to show that additional arguments are optional.
- \textbf{Bold fixed width} font is used to identify brackets that surround optional items in syntax, [], and to identify system output in interactive examples.
- Variables within syntax statements are shown in \textit{italic fixed width font}.
-Ranges of values are indicated with parentheses or brackets as follows:
  — (a,b) means the range of all values from a to b, including neither a nor b
  — [a,b] means the range of all values from a to b, including a and b
  — [a,b) means the range of all values from a to b, including a, but not b
  — (a,b] means the range of all values from a to b, including b, but not a
- Shading is used to identify extensions or warnings as detailed in Section 1.7.1 on page 11.

Notes:
1. Symbolic limits are used in this document instead of fixed values for portability. The values of most of these constants are defined in \texttt{<limits.h>} or \texttt{<unistd.h>}.  
2. The values of errors are defined in \texttt{<errno.h>}.  

Trade Marks

AT&T® is a registered trade mark of AT&T in the U.S.A. and other countries.
HP® is a registered trade mark of Hewlett-Packard.
OSF™ is a trade mark of The Open Software Foundation, Inc.
UNIX® is a registered trade mark in the United States and other countries, licensed exclusively through X/Open Company Limited.
/usr/group® is a registered trade mark of UniForum, the International Network of UNIX System Users.
X/Open™ and the “X” device are trade marks of X/Open Company Limited.
Acknowledgements

X/Open gratefully acknowledges:

• AT&T for permission to reproduce portions of its copyrighted System V Interface Definition (SVID) and material from the UNIX System V Release 2.0 documentation.

• The Institution of Electrical and Electronics Engineers, Inc. for permission to reproduce portions of its copyrighted IEEE Std 1003.2/D12, which have since become the corresponding portions of IEEE Std 1003.2-1992 and ISO/IEC 9945-2:1993, and also for permission to reproduce portions of IEEE Std P1003.1g/D4.

• The IEEE Computer Society’s Portable Applications Standards Committee (PASC), whose Standards contributed to our work.

• The UniForum (formerly /usr/group) Technical Committee’s Internationalization Subcommittee for work on internationalised regular expressions.

• The ANSI X3J11 Committees.

• Hewlett-Packard Company, International Business Machines Corporation, Novell Inc., The Open Software Foundation, and Sun Microsystems, Inc., for their work in developing the Single X/Open UNIX Extension and sponsoring it through the X/Open Direct Review (Fast-track) process.
The following documents are referenced in this specification:

AIX 3.2 Manual
   AIX Version 3.2 For RISC System/6000, Technical Reference: Base Operating System And

ANSI C
   ANSI X3.159-1989, Programming Language C.

ANSI/IEEE Std 754-1985
   Standard for Binary Floating-Point Arithmetic.

ANSI/IEEE Std 854-1987
   Standard for Radix-Independent Floating-Point Arithmetic.

Draft ANSI X3J11.1
   IEEE Floating Point draft report of ANSI X3J11.1 (NCEG).

FIPS 151-2

HP-UX Manual

ISO 4217
   ISO 4217: 1987, Codes for the Representation of Currencies and Funds.

ISO 6937

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

ISO 8859-1
   ISO 8859-1: 1987, Information Processing — 8-bit Single-byte Coded Graphic Character Sets

ISO/IEC 646
   Interchange.

ISO C
   ISO/IEC 9899: 1990, Programming Languages — C (which is technically identical to ANSI
   X3.159-1989, Programming Language C).

ISO POSIX-1
   ISO/IEC 9945-1: 1990, Information Technology — Portable Operating System Interface
   (POSIX) — Part 1: System Application Program Interface (API) [C Language] (which is

ISO POSIX-2
   (POSIX) — Part 2: Shell and Utilities (which is identical to IEEE Std 1003.2-1992).
MSE working draft
Extensions (MSE) as documented in the ISO Working Paper SC22/WG14/N205 dated 31

OSF AES
Application Environment Specification (AES) Operating System Programming Interfaces

OSF/1

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

SunOS 5.3
SunOS 5.3 STREAMS Programmer’s Guide (Part No. 801-5305-10).

SVID Issue 1
System V Interface Definition (Spring 1985 - Issue 1).

SVID Issue 2
System V Interface Definition (Spring 1986 - Issue 2).

SVID 3rd Edition

System V Release 2.0

System V Release 4.2

The following X/Open documents are referenced in this specification.

Curses Interface
X/Open Specification, February 1992, Supplementary Definitions, Issue 3
(ISBN: 1-872630-38-3, C213), Chapters 9 to 14 inclusive, Curses Interface; this specification
was formerly X/Open Portability Guide, Issue 3, Volume 3, January 1989, XSI

Headers Interface
X/Open Specification, February 1992, Supplementary Definitions, Issue 3
(ISBN: 1-872630-38-3, C213), Chapter 19, Cpio and Tar Headers; this specification was
formerly X/Open Portability Guide Issue 3, Volume 3, January 1989, XSI Supplementary

Internationalisation Guide, Version 2

Issue 1

Issue 2
See XSH, Issue 2.
Referenced Documents

Issue 3
See XSH, Issue 3.

Issue 4
See XSH, Issue 4.

Issue 4, Version 2

Migration Guide

XBD, Issue 4

XBD, Issue 4, Version 2

XCU, Issue 4

XCU, Issue 4, Version 2

XNFS

XPG4

XSH, Issue 2

XSH, Issue 3

XSH, Issue 4

XSH, Issue 4, Version 2
1.1 Overview

This document describes the interfaces offered to application programs by the X/Open System Interface (XSI). It defines these interfaces and their run-time behaviour without imposing any particular restrictions on the way in which the interfaces are implemented.

The interfaces are defined in terms of the source code interfaces for the C programming language, which is defined in the ISO C standard. It is possible that some implementations may make the interfaces available to languages other than C, but this document does not currently define the source code interfaces for any other language.

This specification allows an application to be built using a set of services that are consistent across all systems that conform to this specification (see Section 1.2). Such systems are termed XSI-conformant systems. Applications written in C using only these interfaces and avoiding implementation-dependent constructs are portable to all XSI-conformant systems.

1.2 Conformance

An implementation conforming to this document shall meet the requirements specified by BASE conformance (see Section 1.2.1) or by X/OPEN UNIX conformance (see Section 1.2.2 on page 2).

1.2.1 BASE Conformance

An implementation conforming to this document shall meet the following criteria for BASE conformance:

- The system shall support all the interfaces and headers defined within this specification that are part of the BASE capability. The BASE capability includes everything not listed in one of the feature groups defined in Section 1.3 on page 3.
- The system may provide one or more of the following Feature Groups:
  - POSIX2 C-language Binding
  - Shared Memory
  - Encryption
  - Enhanced Internationalisation.
- When an implementation claims that a feature is provided, all of its constituent parts shall be provided and shall comply with this specification.

Note: Each Feature Group can be optional or mandatory as defined in the referenced XPG4 document. Some interfaces in Feature Groups define optional behaviour. To determine whether an implementation supports an optional Feature Group or optional behaviour, refer to the implementation's Conformance Statement.

- The system may provide additional or enhanced interfaces, headers and facilities not required by this specification, provided that such additions or enhancements do not affect the behaviour of an application that requires only the facilities described in this document.
1.2.2 X/Open UNIX Conformance

This document (from Issue 4, Version 2) is one of the specifications that contribute to the definition of X/Open UNIX. All UNIX implementations shall conform to this document according to the following criteria:

- The system shall support all the interfaces and headers defined within this specification that are part of the BASE capability as defined above.

- The system shall support all the following Feature Groups defined within this specification:
  — POSIX2 C-language Binding
  — Shared Memory
  — Enhanced Internationalisation
  — X/Open UNIX Extension.

- The system may support the following Feature Group defined within this specification:
  — Encryption.

- When an implementation claims that a feature is provided, all of its constituent parts shall be provided and shall comply with this specification.

**Note:** The Encryption Feature Group can be optional or mandatory as defined in the referenced XPG4 document. Some interfaces in this Feature Group define optional behaviour. To determine whether an implementation supports this Feature Group, refer to the implementation's Conformance Statement.

- The system may provide additional or enhanced interfaces, headers and facilities not required by this specification, provided that such additions or enhancements do not affect the behaviour of an application that requires only the facilities described in this document.
1.3 Feature Groups

The interfaces to elements of Feature Groups other than the X/Open UNIX EXTENSION shall exist on all implementations; however, on implementations that do not support these interfaces, each interface shall indicate an error, with _errno_ set to [ENOSYS] unless otherwise specified.

1.3.1 POSIX2 C-language Binding

The POSIX2 C-language Binding feature includes the following interfaces:

```c
fnmatch()
glob()
globfree()
regcomp()
regerror()
regexec()
regfree()
wordexp()
wordfree()
```

These are identified as POSIX2 CLB at the tops of applicable pages.

An implementation that claims conformance to this Feature Group shall set [_POSIX2_C_VERSION] to the value specified in `<unistd.h>` on page 855. An implementation that does not claim conformance to this Feature Group shall set [_POSIX2_C_VERSION] to −1.

1.3.2 Shared Memory

The Shared Memory feature includes the following interfaces:

```c
shmat()
shmctl()
shmdt()
shmget()
```

These are identified as SHARED MEM at the tops of applicable pages.

The Shared Memory functions allow different processes to share access to the same data, by having the memory on which that data is stored appear in each process’ address space. Some existing hardware architectures or operating system models are unsuitable for support of these functions.

An implementation that claims conformance to this Feature Group shall set [_XOPEN_SHM] to a value other than −1. An implementation that does not claim conformance to this Feature Group shall set [_XOPEN_SHM] to −1.

1.3.3 Encryption

The Encryption feature includes the following interfaces:

```c
crypt()
encrypt()
setkey()
```

These are identified as CRYPT at the tops of applicable pages.
Due to U.S. Government export restrictions on the decoding algorithm, implementations are restricted in making these functions available. All the functions in the Encryption Feature Group may therefore return [ENOSYS] or alternatively, `encrypt()` shall return [ENOSYS] for the decryption operation.

An implementation that claims conformance to this Feature Group shall set `_XOPEN_CRYPT` to a value other than −1. An implementation that does not claim conformance to this Feature Group shall set `_XOPEN_CRYPT` to −1.

### 1.3.4 Enhanced Internationalisation

The Enhanced Internationalisation feature includes the following interfaces:

- `strfmon()`
- `strptime()`
- `wcscoll()`
- `wcsftime()`
- `wcsxfrm()`

These are identified as ENHANCED I18N at the tops of applicable pages.

These interfaces are new. Their implementations are not yet in common usage, nor are they consistent. They are included to encourage the convergence of implementations. They will be made part of the BASE capability in a future issue of this document.

An implementation that claims conformance to this Feature Group shall set `_XOPEN_ENH_I18N` to a value other than −1. An implementation that does not claim conformance to this Feature Group shall set `_XOPEN_ENH_I18N` to −1.

### 1.3.5 X/Open UNIX Extension

This document includes interfaces that had not previously been adopted by X/Open. These are now included to provide portability for applications originally written to be compiled on UNIX and UNIX-based operating systems.

Where entire manual pages are added, the legend X/OPEN UNIX appears at the top of each page. Where additional semantics are added to existing manual pages, the new material is identified by use of the UX margin legend.

An implementation that does not claim conformance to this Feature Group will either set `_XOPEN_UNIX` to −1 (the preferred option) or leave it undefined (pre-dates the introduction of this Feature Group).
1.4 Changes from Issue 3
The following sections describe changes made to this document since Issue 3. The CHANGE HISTORY section for each entry details the technical changes that have been made to that entry since Issue 3. Changes made between Issue 2 and Issue 3 are not included.

1.4.1 Changes from Issue 3 to Issue 4
The following list summarises the major changes that were made in this document from Issue 3 to Issue 4:

• Curses interface functions are not specified in this document but in the Curses Interface specification.
• All functionality has been aligned with the relevant standards, see Section 1.6 on page 10.
• The function definitions use the ISO C syntax.
• Some additional functions and headers are included; these are listed in the table in Section 1.4.3 on page 6.
• World-wide Portability Interfaces have been added.

1.4.2 Changes from Issue 4 to Issue 4, Version 2
The following list summarises the major changes that have been made in this document since Issue 4:

• The X/Open UNIX extension has been added. This specifies the common core APIs of 4.3 Berkeley Software Distribution (BSD 4.3), the OSF AES and SVID Issue 3.
• STREAMS have been added as part of the X/Open UNIX extension.
• Existing XPG4 interfaces have been clarified as a result of industry feedback.

Note: A new specification is currently under development covering internationalised curses.
## 1.4.3 New Features

The interfaces, headers and external variables first introduced in Issue 4, Version 2 are listed in the table below.

<table>
<thead>
<tr>
<th>New Interfaces, Headers and External Variables in Issue 4, Version 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>FD_CLR()</td>
</tr>
<tr>
<td>FD_ISSET()</td>
</tr>
<tr>
<td>FD_SET()</td>
</tr>
<tr>
<td>FD_ZERO()</td>
</tr>
<tr>
<td>_longjmp()</td>
</tr>
<tr>
<td>_setjmp()</td>
</tr>
<tr>
<td>a64l()</td>
</tr>
<tr>
<td>acosh()</td>
</tr>
<tr>
<td>asinh()</td>
</tr>
<tr>
<td>atanh()</td>
</tr>
<tr>
<td>bcopy()</td>
</tr>
<tr>
<td>dbm_close()</td>
</tr>
<tr>
<td>dbm_clearerr()</td>
</tr>
<tr>
<td>dbm_close()</td>
</tr>
<tr>
<td>dbm_delete()</td>
</tr>
<tr>
<td>dbm_error()</td>
</tr>
<tr>
<td>dbm_fetch()</td>
</tr>
<tr>
<td>dbm_firstkey()</td>
</tr>
<tr>
<td>dbm_nextkey()</td>
</tr>
<tr>
<td>dbm_open()</td>
</tr>
<tr>
<td>dbm_store()</td>
</tr>
<tr>
<td>dirname()</td>
</tr>
<tr>
<td>ecvt()</td>
</tr>
<tr>
<td>endgrent()</td>
</tr>
<tr>
<td>endpwent()</td>
</tr>
<tr>
<td>&lt;fmtmsg.h&gt;</td>
</tr>
<tr>
<td>&lt;libgen.h&gt;</td>
</tr>
<tr>
<td>&lt;ndbm.h&gt;</td>
</tr>
<tr>
<td>&lt;poll.h&gt;</td>
</tr>
<tr>
<td>getdate_err</td>
</tr>
</tbody>
</table>
The interfaces and headers first introduced in Issue 4 are listed in the table below.

<table>
<thead>
<tr>
<th>New Interfaces and Headers in Issue 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>atexit()</code></td>
</tr>
<tr>
<td><code>fgetpos()</code></td>
</tr>
<tr>
<td><code>fputwc()</code></td>
</tr>
<tr>
<td><code>getwchar()</code></td>
</tr>
<tr>
<td><code>iconv_close()</code></td>
</tr>
<tr>
<td><code>iswctr1()</code></td>
</tr>
<tr>
<td><code>iswlower()</code></td>
</tr>
<tr>
<td><code>iswupper()</code></td>
</tr>
<tr>
<td><code>localeconv()</code></td>
</tr>
<tr>
<td><code>memmove()</code></td>
</tr>
<tr>
<td><code>regcomp()</code></td>
</tr>
<tr>
<td><code>strftime()</code></td>
</tr>
<tr>
<td><code>towlower()</code></td>
</tr>
<tr>
<td><code>wcschr()</code></td>
</tr>
<tr>
<td><code>wcscspn()</code></td>
</tr>
<tr>
<td><code>wcsncmp()</code></td>
</tr>
<tr>
<td><code>wcsspn()</code></td>
</tr>
<tr>
<td><code>wcsrmb()</code></td>
</tr>
<tr>
<td><code>wcsxfm()</code></td>
</tr>
<tr>
<td><code>wordexp()</code></td>
</tr>
</tbody>
</table>

The functions `msgctl()`, `msgget()`, `msgrec()`, `msgsnd()`, `semctl()`, `semget()` and `semop()` had OPTIONAL FUNCTIONALITY in XPG3, but are part of the BASE capability in this document, and are therefore mandatory.

The headers `<cpio.h>` and `<tar.h>` are included above only because in XPG3 they appeared in Volume 3, rather than Volume 2.
1.4.4 To Be Withdrawn

Some of the interfaces and headers in this issue are marked TO BE WITHDRAWN. Various factors may have contributed to the decision to withdraw an interface. In all cases, the reasons for withdrawal of an interface are documented on the relevant pages.

If a migration path exists, advice is given to application developers regarding alternative means of obtaining similar functionality. This information may be found in the APPLICATION USAGE sections on the relevant pages.

Interfaces marked TO BE WITHDRAWN shall still exist on conformant implementations. However, they will be marked WITHDRAWN in a future issue of this document. Interfaces marked WITHDRAWN may still exist on conformant implementations.

Application writers should not use functionality marked TO BE WITHDRAWN.

The following interfaces, headers and external variables are marked TO BE WITHDRAWN in this document:

<table>
<thead>
<tr>
<th>Interfaces, Headers and External Variables To Be Withdrawn</th>
</tr>
</thead>
<tbody>
<tr>
<td>advance()</td>
</tr>
<tr>
<td>getpass()</td>
</tr>
<tr>
<td>sigstack()</td>
</tr>
<tr>
<td>&lt;regexp.h&gt;</td>
</tr>
<tr>
<td>getdate_err</td>
</tr>
</tbody>
</table>

1.4.5 Withdrawn

No interfaces or headers in this issue are marked WITHDRAWN. Two constants are marked WITHDRAWN from <limits.h> because they have been moved to <float.h> and their definitions have been changed. The ENOTBLK constant has been withdrawn from <errno.h>.

Withdrawn functionality may still exist on conformant implementations. However, they will not appear in the next issue of this document. Application writers should not use functionality marked WITHDRAWN.

Constants marked in this issue as WITHDRAWN are:

<table>
<thead>
<tr>
<th>Withdrawn Constants</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBL_MIN</td>
</tr>
</tbody>
</table>
1.5 Terminology

The following terms are used in this specification:

**can**
This describes a permissible optional feature or behaviour available to the user or application; all systems support such features or behaviour as mandatory requirements.

**implementation-dependent**
The value or behaviour is not consistent across all implementations. The provider of an implementation normally documents the requirements for correct program construction and correct data in the use of that value or behaviour. When the value or behaviour in the implementation is designed to be variable or customisable on each instantiation of the system, the provider of the implementation normally documents the nature and permissible ranges of this variation. Applications that are intended to be portable must not rely on implementation-dependent values or behaviour.

**may**
With respect to implementations, the feature or behaviour is optional. Applications should not rely on the existence of the feature. To avoid ambiguity, the reverse sense of *may* is expressed as *need not*, instead of *may not*.

**must**
This describes a requirement on the application or user.

**obsolescent**
Certain features are *obsolescent*, which means that they may be considered for withdrawal in future revisions of this document. They are retained in this version because of their widespread use. Their use in new applications is discouraged.

**should**
With respect to implementations, the feature is recommended, but it is not mandatory. Applications should not rely on the existence of the feature.

With respect to users or applications, the word means recommended programming practice that is necessary for maximum portability.

**undefined**
A value or behaviour is undefined if this document imposes no portability requirements on applications for erroneous program constructs or erroneous data. Implementations may specify the result of using that value or causing that behaviour, but such specifications are not guaranteed to be consistent across all implementations. An application using such behaviour is not fully portable to all systems.

**unspecified**
A value or behaviour is unspecified if this document imposes no portability requirements on applications for correct program construct or correct data. Implementations may specify the result of using that value or causing that behaviour, but such specifications are not guaranteed to be consistent across all implementations. An application requiring a specific behaviour, rather than tolerating any behaviour when using that functionality, is not fully portable to all systems.

**will**
This means that the behaviour described is a requirement on the implementation and applications can rely on its existence.
1.6 **Relationship to Formal Standards**

Great care has been taken to ensure that this document is fully aligned with the following formal standards:

- ISO/IEC 9945-1: 1990 (ISO POSIX-1)
- ISO/IEC 9899: 1990 (ISO C)
- Federal Information Procurement Standards (FIPS) 151-2.

Any conflict between this document and any of these standards is unintentional. This document defers to the formal standards, which X/Open recognises as superior. In particular, from time to time, when ambiguities are found in the formal standards, the responsible bodies will make interpretations of them, whose findings become binding on the standard. Where, as the result of such an interpretation, or for any other reason, any of these formal standards are found to conflict with this document, XSI-conformant systems are required to behave in the manner defined either by the formal standard or by this document. Application writers should clearly avoid depending exclusively on either behaviour in such cases; the list of all conflicts found since publication of this document is available on request. (See page ii for how to contact X/Open.)

This document also allows, but does not require, mathematics functions to support IEEE Std 754-1985 and IEEE Std 854-1987.

1.6.1 **Relationship to Emerging Formal Standards**

This document has also been aligned with the important, but not yet ratified formal standard:


A later edition of this document will be fully aligned with that standard.

This document also allows, but does not require, mathematics functions to behave as specified by the IEEE Floating Point draft report of ANSI X3J11.1 (NCEG).

Where function specifications in the draft ANSI X3J11.1 require behaviour that is different from this document, but not in conflict with the ISO C standard, an XSI-conformant system may behave either in the manner defined by the draft ANSI X3J11.1 or by this document.
1.7 Portability

This document describes a superset of the requirements of the ISO POSIX-1 standard and the ISO C standard. It also contains parts of the ISO POSIX-2 standard Shell and Utilities which X/Open feels are better suited to inclusion in this document, rather than in the XCU specification. (The ISO POSIX-1 standard is identical to IEEE Std 1003.1-1990, which is often referred to as the POSIX.1 standard. The ISO C standard is technically identical in normative content to the ANSI C standard.)

Some of the utilities in X/Open CAE Specification, Commands and Utilities, Issue 4, Version 2 and functions in this document describe functionality that might not be fully portable to systems based on the ISO POSIX-2 standard or the ISO POSIX-1 standard. Where enhanced or reduced functionality is specified, the text is shaded and a code in the margin identifies the nature of the extension or warning (see Section 1.7.1). For maximum portability, an application should avoid such functionality.

1.7.1 Codes

The codes and their meanings are as follows:

EI  Enhanced internationalisation.
    This identifies the interfaces in the Enhanced Internationalisation Feature Group in this document.

EX  Extension.
    The functionality described is an extension to the standards referenced above. Application writers may confidently make use of an extension as it will be supported on all XSI-conformant systems. These extensions are designed not to conflict with the published standards.

    If an entire SYNOPSIS section is shaded and marked with one EX, all the functionality described in that entry is an extension.

    Some behaviour which is allowed to be optional in the formal standards is mandated on XSI-conformant systems. Such behaviours (for example, those dependent on the availability of job control) may not be individually marked as extensions, but the mandatory nature of the feature is marked as an extension where the option is described, typically in the header file where the corresponding symbolic constant is defined.

FIPS  FIPS Extension.
    The Federal Information Processing Standards (FIPS) are a series of U.S. government procurement standards managed and maintained on behalf of the U.S. Department of Commerce by the National Institute of Standards and Technology (NIST). Where extensions have been made in order to align with the FIPS requirements, they have the special mark shown here, and appear in the index under FIPS alignment (as well as under EX).

    The following extensions are required by FIPS 151-2:

    • The implementation will support {POSIX_CHOWN_RESTRICTED}.
    • The limit {NGROUPS_MAX} will be greater than or equal to 8.
    • The implementation will support the setting of the group ID of a file (when it is created) to that of the parent directory.
    • The implementation will support {POSIX_SAVED_IDS}.
    • The implementation will support {POSIX_VDISABLE}.
    • The implementation will support {POSIX_JOB_CONTROL}.
Portability Introduction

- The implementation will support \_POSIX\_NO\_TRUNC.
- The read() call returns the number of bytes read when interrupted by a signal and will not return -1.
- The write() call returns the number of bytes written when interrupted by a signal and will not return -1.
- In the environment for the login shell, the environment variables \_LOGNAME\_ and \_HOME\_ will be defined and have the properties described in Chapter 5 of X/Open CAE Specification, System Interface Definitions, Issue 4, Version 2.
- The value of \_CHILD\_MAX\_ will be greater than or equal to 25.
- The value of \_OPEN\_MAX\_ will be greater than or equal to 20.
- The implementation will support the functionality associated with the symbols CS7, CS8, CSTOPO, PARODD and PARENb defined in <termios.h>.

**Optional header.**

In the SYNOPSIS section of some interfaces in this document an included header is marked as in the following example:

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

This indicates that the marked header is not required on XSI-conformant systems. This is an extension to certain formal standards where the full synopsis is required.

**X/Open UNIX Extension**

The material relates to interfaces included to provide portability for applications originally written to be compiled on UNIX and UNIX-based operating systems. Therefore, the features described may not be present on systems that conform to XPG4 or to earlier XPG releases. The relevant reference manual pages may provide additional or more specific portability warnings about use of the material.

If an entire SYNOPSIS section is shaded and marked with one UX, all the functionality described in that entry is an extension.

The material on pages labelled X/OPEN UNIX and the material flagged with the UX margin legend is available only in cases where the _XOPEN_UNIX_ version test macro is defined.

**World-wide portability extension.**

These interfaces form part of the set of World-wide Portability (WP) interfaces that provide additional support for the internationalisation of applications.

If an entire SYNOPSIS section is marked with WP, this means that all the functionality described in that entry is part of this internationalisation support.

These WP interfaces extend this document to provide support for multiple byte code sets and thus potentially all national languages not previously supportable within, for example, 8-bit code sets. The WP interfaces are aligned with the working draft of ISO/IEC 9899:1990/Add.3: draft, Addendum 3 - Multibyte Support Extensions (MSE) as documented in the ISO Working Paper SC22/WG14/N205 dated 31 March 1992.

The Internationalisation Guide contains specific information on the internationalisation of applications.
1.8 Format of Entries

The entries in Chapter 3 and Chapter 4 are based on a common format.

NAME
This section gives the name or names of the entry and briefly states its purpose.

SYNOPSIS
This section summarises the use of the entry being described. If it is necessary to include a header to use this interface, the names of such headers are shown, for example:

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

DESCRIPTION
This section describes the functionality of the interface or header.

RETURN VALUE
This section indicates the possible return values, if any.

If the implementation can detect errors, “successful completion” means that no error has been detected during execution of the function. If the implementation does detect an error, the error will be indicated.

For functions where no errors are defined, “successful completion” means that if the implementation checks for errors, no error has been detected. If the implementation can detect errors, and an error is detected, the indicated return value will be returned and `errno` may be set.

ERRORS
This section gives the symbolic names of the values returned in `errno` if an error occurs.

“No errors are defined” means that values and usage of `errno`, if any, depend on the implementation.

EXAMPLES
This section gives examples of usage, where appropriate.

APPLICATION USAGE
This section gives warnings and advice to application writers about the entry.

FUTURE DIRECTIONS
This section provides comments which should be used as a guide to current thinking; there is not necessarily a commitment to adopt these future directions.

SEE ALSO
This section gives references to related information.

CHANGE HISTORY
This section shows the derivation of the entry and any significant changes that have been made to it.

The only sections relating to conformance are the SYNOPSIS, DESCRIPTION, RETURN VALUE and ERRORS sections.
Introduction
Chapter 2

General Information

This chapter covers information that is relevant to all the Interfaces specified in Chapter 3 and Chapter 4:

- the use and implementation of interfaces (see Section 2.1)
- the compilation environment (see Section 2.2 on page 17)
- error numbers (see Section 2.3 on page 25)
- standard I/O streams (see Section 2.4 on page 32)
- STREAMS (see Section 2.5 on page 35)
- interprocess communication (IPC) (see Section 2.6 on page 37)
- data types (see Section 2.7 on page 38).

2.1 Use and Implementation of Interfaces

Each of the following statements applies unless explicitly stated otherwise in the detailed descriptions that follow. If an argument to a function has an invalid value (such as a value outside the domain of the function, or a pointer outside the address space of the program, or a null pointer), the behaviour is undefined. Any function declared in a header may also be implemented as a macro defined in the header, so a library function should not be declared explicitly if its header is included. Any macro definition of a function can be suppressed locally by enclosing the name of the function in parentheses, because the name is then not followed by the left parenthesis that indicates expansion of a macro function name. For the same syntactic reason, it is permitted to take the address of a library function even if it is also defined as a macro. The use of the C-language `#undef` construct to remove any such macro definition will also ensure that an actual function is referred to. Any invocation of a library function that is implemented as a macro will expand to code that evaluates each of its arguments exactly once, fully protected by parentheses where necessary, so it is generally safe to use arbitrary expressions as arguments. Likewise, those function-like macros described in the following sections may be invoked in an expression anywhere a function with a compatible return type could be called.

Provided that a library function can be declared without reference to any type defined in a header, it is also permissible to declare the function, either explicitly or implicitly, and use it without including its associated header. If a function that accepts a variable number of arguments is not declared (explicitly or by including its associated header), the behaviour is undefined.

As a result of changes in this issue of this document, application writers are only required to include the minimum number of headers. Implementations of XSI-conformant systems will make all necessary symbols visible as described in the Headers section of this document.
2.1.1 C Language in an Issue 4 Environment

ISO C is the language specified in the referenced ISO C standard. Common Usage C refers to the C language before standardisation. Differences exist between ISO C and Common Usage C. Issues 1 to 3 of the X/Open Portability Guide define Common Usage C as the XSI C language, whereas Issue 4 defines ISO C as the XSI C language.

Differences also exist between the C library definitions published in successive issues of the X/Open Portability Guide. Issues 1 and 2 derived their C library definitions from Issue 1 of the SVID; Issue 3 retains the same function syntax as Issue 2 but header usage is aligned with the library definition published in the draft ANS X3.159 standard. Issue 4 is aligned with the library definition published in the ISO C standard.

Information about the portability of applications between Issues 3 and 4 is contained in this document and in the Migration Guide, in terms of language and C library usage. In particular, the CHANGE HISTORY sections in this document show where the type of arguments and return values of individual functions are changed to align with ISO C.

Application programmers are warned that the use of earlier definitions of C library functions may not be fully supported in Issue 4 environments. Information about migration to the use of ANSI C headers can be found in the APPLICATION USAGE and CHANGE HISTORY sections of Issue 3. Much of this guidance is equally relevant to Issue 4 environments.

2.1.2 Use of File System Interfaces

The Interfaces in this volume that operate on files can behave differently if the file that is being operated on has been made available by a network file system. If the network file system is an XSI-conformant system conforming to the XNFS specification, the differences that can occur are detailed in Appendices A and B of that document.
2.2 The Compilation Environment

Applications should ensure that the feature test macro \_XOPEN\_SOURCE is defined before inclusion of any header. This is needed to enable the functionality described in this document (but see also Section 2.2.1), and possibly to enable functionality defined elsewhere in the Common Applications Environment.

The \_XOPEN\_SOURCE macro may be defined automatically by the compilation process, but to ensure maximum portability, applications should make sure that \_XOPEN\_SOURCE is defined by using either compiler options or \#define directives in the source files, before any \#include directives. Identifiers in this document may only be undefined using the \#undef directive as described in Section 2.1 on page 15 or Section 2.2.2. These \#undef directives must follow all \#include directives of any XSI headers.

Most strictly conforming POSIX and ISO C applications will compile on systems compliant to this specification. However, an application which uses any of the items marked as an extension to POSIX and ISO C, for any purpose other than that shown here, may not compile. In such cases, it may be necessary to alter those applications to use alternative identifiers.

Since this document is aligned with the ISO C standard, and since all functionality enabled by the \_POSIX\_C\_SOURCE set equal to 2 should be enabled by \_XOPEN\_SOURCE, there should be no need to define either \_POSIX\_SOURCE or \_POSIX\_C\_SOURCE if \_XOPEN\_SOURCE is defined. Therefore if \_XOPEN\_SOURCE is defined and \_POSIX\_SOURCE is defined, or \_POSIX\_C\_SOURCE is set equal to 1 or 2, the behaviour is the same as if only \_XOPEN\_SOURCE is defined. However should \_POSIX\_C\_SOURCE be set to a value greater than 2, the behaviour is undefined.

2.2.1 X/Open UNIX Extensions

An application that uses any function specified as X/Open UNIX or relies on any portion of an X/Open specification marked with the UX margin legend must define \_XOPEN\_SOURCE_EXTENDED = 1 in each source file or as part of its compilation environment, in addition to \_XOPEN\_SOURCE. When \_XOPEN\_SOURCE_EXTENDED = 1 is defined in a source file, it must appear before any header is included. The compilation environment will not automatically define the \_XOPEN\_SOURCE_EXTENDED macro.

2.2.2 The X/Open Name Space

All identifiers in this document except environ are defined in at least one of the headers, as shown in Chapter 4. When \_XOPEN\_SOURCE is defined, each header defines or declares some identifiers, potentially conflicting with identifiers used by the application. The set of identifiers visible to the application consists of precisely those identifiers from the header pages of the included headers, as well as additional identifiers reserved for the implementation. In addition, some headers may make visible identifiers from other headers as indicated on the relevant header pages.

The identifiers reserved for use by the implementation are described below.

1. Each identifier with external linkage described in the header section is reserved for use as an identifier with external linkage if the header is included.

2. Each macro name described in the header section is reserved for any use if the header is included.

3. Each identifier with file scope described in the header section is reserved for use as an identifier with file scope in the same name space if the header is included.
4. Identifiers with the prefixes, suffixes or complete names marked UX in the following tables are reserved for use by the implementation only if the application has defined 

```c
#define _XOPEN_SOURCE_EXTENDED 1
```
If any header in the following table is included, identifiers with the prefixes, suffixes or complete names shown are reserved for any use by the implementation.

<table>
<thead>
<tr>
<th>Header</th>
<th>Prefix</th>
<th>Suffix</th>
<th>Complete Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;arpa/inet.h&gt;</code></td>
<td>in_</td>
<td>inet_</td>
<td></td>
</tr>
<tr>
<td><code>&lt;dirent.h&gt;</code></td>
<td>d_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;errno.h&gt;</code></td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;fcntl.h&gt;</code></td>
<td>l_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;glob.h&gt;</code></td>
<td>gl_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;grp.h&gt;</code></td>
<td>gr_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;limits.h&gt;</code></td>
<td></td>
<td>_MAX</td>
<td></td>
</tr>
<tr>
<td><code>&lt;locale.h&gt;</code></td>
<td>LC_</td>
<td>[A-Z]</td>
<td></td>
</tr>
<tr>
<td><code>&lt;ndbm.h&gt;</code></td>
<td>dbm_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;netdb.h&gt;</code></td>
<td>h_, n_, p_, s_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;netinet/in.h&gt;</code></td>
<td>in_, s_, sin_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;poll.h&gt;</code></td>
<td>pd_, ph_, ps_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;pwd.h&gt;</code></td>
<td>pw_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;re_comp.h&gt;</code></td>
<td>re_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;regex.h&gt;</code></td>
<td>re_, rm_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;regexp.h&gt;</code></td>
<td>re_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;signal.h&gt;</code></td>
<td>sa_, SIG[A-Z], SIG_[A-Z]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;stropts.h&gt;</code></td>
<td>si_, ss_, sv_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;sys/ipc.h&gt;</code></td>
<td>ipc_</td>
<td></td>
<td>key, pad, seq</td>
</tr>
<tr>
<td><code>&lt;sys/msg.h&gt;</code></td>
<td>msg</td>
<td></td>
<td>msg</td>
</tr>
<tr>
<td><code>&lt;sys/resource.h&gt;</code></td>
<td>rlim_, ru_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;sys/sems.h&gt;</code></td>
<td>sem</td>
<td></td>
<td>sem</td>
</tr>
<tr>
<td><code>&lt;sys/shm.h&gt;</code></td>
<td>shm</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;sys/socket.h&gt;</code></td>
<td>sa_, if_, ifc_, ifru_, infu_, ifra_, msg_, cmsg_, l_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;sys/stat.h&gt;</code></td>
<td>st_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;sys/statvfs.h&gt;</code></td>
<td>f_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;sys/time.h&gt;</code></td>
<td>fds_, it_, tv_, FD_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;sys/times.h&gt;</code></td>
<td>tms_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;sys/uio.h&gt;</code></td>
<td>iov_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;sys/un.h&gt;</code></td>
<td>sun_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;sys/utsname.h&gt;</code></td>
<td>uts_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;sys/wait.h&gt;</code></td>
<td>si_, W[A-Z]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;termios.h&gt;</code></td>
<td>c_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;time.h&gt;</code></td>
<td>tm_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;ucontext.h&gt;</code></td>
<td>uc_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;ulimit.h&gt;</code></td>
<td>UL_</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;utmpx.h&gt;</code></td>
<td>ut_</td>
<td>_LVL, _TIME, _PROCESS</td>
<td></td>
</tr>
<tr>
<td><code>&lt;wordexp.h&gt;</code></td>
<td>we_</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The notation [A-Z] indicates any upper-case letter in the portable character set. The notation [a-z] indicates any lower-case letter in the portable character set. Commas and spaces in the lists of prefixes and complete names in the above table are not part of any prefix or complete name.
For historical compatibility, `<regexp.h>` is allowed to contain any additional symbol that does not interfere with other interfaces in this specification.

If any header in the following table is included, macros with the prefixes shown may be defined. After the last inclusion of a given header, an application may use identifiers with the corresponding prefixes for its own purpose, provided their use is preceded by an `#undef` of the corresponding macro.

<table>
<thead>
<tr>
<th>Header</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;curses.h&gt;</code></td>
<td>A_ ACS_ COLOR_ KEY_ WA_</td>
</tr>
<tr>
<td><code>&lt;fcntl.h&gt;</code></td>
<td>F_ O_ S_</td>
</tr>
<tr>
<td><code>&lt;fmtmsg.h&gt;</code></td>
<td>MM_</td>
</tr>
<tr>
<td><code>&lt;fnmatch.h&gt;</code></td>
<td>FNM_</td>
</tr>
<tr>
<td><code>&lt;ftw.h&gt;</code></td>
<td>FTW</td>
</tr>
<tr>
<td><code>&lt;glob.h&gt;</code></td>
<td>GLOB_</td>
</tr>
<tr>
<td><code>&lt;ndbm.h&gt;</code></td>
<td>DBM_</td>
</tr>
<tr>
<td><code>&lt;netdb.h&gt;</code></td>
<td>NO_</td>
</tr>
<tr>
<td><code>&lt;netinet/in.h&gt;</code></td>
<td>IMPLINK_ IN_ INADDR_ IP_ IPPORT_</td>
</tr>
<tr>
<td><code>&lt;nl_types.h&gt;</code></td>
<td>NL_</td>
</tr>
<tr>
<td><code>&lt;poll.h&gt;</code></td>
<td>POLL</td>
</tr>
<tr>
<td><code>&lt;re_comp.h&gt;</code></td>
<td>REG_</td>
</tr>
<tr>
<td><code>&lt;regex.h&gt;</code></td>
<td>REG_</td>
</tr>
<tr>
<td><code>&lt;signal.h&gt;</code></td>
<td>BUS_ CLD_ FPE_ ILL_ POLL_</td>
</tr>
<tr>
<td><code>&lt;stropts.h&gt;</code></td>
<td>FLUSH[A-Z] I_ SND[A-Z]</td>
</tr>
<tr>
<td><code>&lt;syslog.h&gt;</code></td>
<td>LOG_</td>
</tr>
<tr>
<td><code>&lt;sys/ipc.h&gt;</code></td>
<td>IPC_</td>
</tr>
<tr>
<td><code>&lt;sys/msg.h&gt;</code></td>
<td>MSG[A-Z] MSG_[A-Z]</td>
</tr>
<tr>
<td><code>&lt;sys/resource.h&gt;</code></td>
<td>PRIO_ RLIM_ RLIMIT_ RUSAGE_</td>
</tr>
<tr>
<td><code>&lt;sys/sem.h&gt;</code></td>
<td>SEM_</td>
</tr>
<tr>
<td><code>&lt;sys/shm.h&gt;</code></td>
<td>SHM[A-Z] SHM_[A-Z]</td>
</tr>
<tr>
<td><code>&lt;sys/socket.h&gt;</code></td>
<td>AF_ MSG_ PF_ SO</td>
</tr>
<tr>
<td><code>&lt;sys/stat.h&gt;</code></td>
<td>S_</td>
</tr>
<tr>
<td><code>&lt;sys/statvfs.h&gt;</code></td>
<td>ST_</td>
</tr>
<tr>
<td><code>&lt;sys/time.h&gt;</code></td>
<td>FD_ ITIMER_</td>
</tr>
<tr>
<td><code>&lt;sys/uio.h&gt;</code></td>
<td>IOV_</td>
</tr>
<tr>
<td><code>&lt;sys/wait.h&gt;</code></td>
<td>P_ BUS_ CLD_ FPE_ TRAP_ ILL_ POLL_</td>
</tr>
<tr>
<td><code>&lt;termios.h&gt;</code></td>
<td>V I O TC B[0-9]</td>
</tr>
<tr>
<td><code>&lt;wordexp.h&gt;</code></td>
<td>WRDE_</td>
</tr>
</tbody>
</table>

**Note:** The notation `[0-9]` indicates any digit. The notation `[A-Z]` indicates any upper-case letter in the portable character set. The notation `[0-9a-z_]` indicates any digit, any lower-case letter in the portable character set or underscore.
If any of the headers referenced by the X/Open **Transport Interface (XTI)** specification are included, then all of the following are reserved:

<table>
<thead>
<tr>
<th>Header</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;xti.h&gt;</code></td>
<td>1_t_ INET_IP_ISO_OPT_T_ TCL_TCP_TCO_XTI_</td>
</tr>
</tbody>
</table>
The following identifiers are reserved regardless of the inclusion of headers.

1. All identifiers that begin with an underscore and either an upper-case letter or another underscore are always reserved for any use by the implementation.

2. All identifiers that begin with an underscore are always reserved for use as identifiers with file scope in both the ordinary identifier and tag name spaces.

3. All identifiers in the table below are reserved for use as identifiers with external linkage. Some of these identifiers do not appear in this document, but are reserved for future use by the ISO C standard.

<table>
<thead>
<tr>
<th>abort</th>
<th>cos</th>
<th>floorl</th>
<th>ldexp</th>
<th>printf</th>
<th>sqrtl</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>cosf</td>
<td>fmod</td>
<td>ldexpf</td>
<td>putc</td>
<td>srand</td>
</tr>
<tr>
<td>acos</td>
<td>cosh</td>
<td>fmodf</td>
<td>ldexp1</td>
<td>putchar</td>
<td>sscanf</td>
</tr>
<tr>
<td>acosf</td>
<td>coshf</td>
<td>fmodl</td>
<td>ldiv</td>
<td>puts</td>
<td>str[a-z]*</td>
</tr>
<tr>
<td>acosl</td>
<td>coshl</td>
<td>fopen</td>
<td>localeconv</td>
<td>qsort</td>
<td>system</td>
</tr>
<tr>
<td>asctime</td>
<td>cosl</td>
<td>fprintf</td>
<td>localtime</td>
<td>raise</td>
<td>tan</td>
</tr>
<tr>
<td>asin</td>
<td>ctime</td>
<td>fputc</td>
<td>log</td>
<td>rand</td>
<td>tanf</td>
</tr>
<tr>
<td>asinf</td>
<td>difftime</td>
<td>fputs</td>
<td>log10</td>
<td>realloc</td>
<td>tanh</td>
</tr>
<tr>
<td>asinl</td>
<td>div</td>
<td>fread</td>
<td>log10f</td>
<td>remove</td>
<td>tanhf</td>
</tr>
<tr>
<td>atan</td>
<td>errno</td>
<td>free</td>
<td>log10l</td>
<td>rename</td>
<td>tanhl</td>
</tr>
<tr>
<td>atan2</td>
<td>exit</td>
<td>freopen</td>
<td>logf</td>
<td>rewind</td>
<td>tanl</td>
</tr>
<tr>
<td>atan2f</td>
<td>exp</td>
<td>frexp</td>
<td>log1</td>
<td>scanf</td>
<td>time</td>
</tr>
<tr>
<td>atan2l</td>
<td>expf</td>
<td>frexpf</td>
<td>longjmp</td>
<td>setbuf</td>
<td>tmpfile</td>
</tr>
<tr>
<td>atanf</td>
<td>expl</td>
<td>frexpl</td>
<td>malloc</td>
<td>setjmp</td>
<td>tmppnam</td>
</tr>
<tr>
<td>atanl</td>
<td>fabs</td>
<td>fscanf</td>
<td>mblen</td>
<td>setlocale</td>
<td>to[a-z]*</td>
</tr>
<tr>
<td>atexit</td>
<td>fabsf</td>
<td>fseek</td>
<td>mbstowcs</td>
<td>setvbuf</td>
<td>ungetc</td>
</tr>
<tr>
<td>atof</td>
<td>fabsl</td>
<td>fsetpos</td>
<td>mbtowc</td>
<td>signal</td>
<td>va_end</td>
</tr>
<tr>
<td>atoi</td>
<td>fclose</td>
<td>ftell</td>
<td>mem[a-z]*</td>
<td>sin</td>
<td>vfprintf</td>
</tr>
<tr>
<td>atol</td>
<td>feof</td>
<td>fwrite</td>
<td>mktime</td>
<td>sinc</td>
<td>vprintf</td>
</tr>
<tr>
<td>bsearch</td>
<td>ferror</td>
<td>getc</td>
<td>modf</td>
<td>sinh</td>
<td>vsprintf</td>
</tr>
<tr>
<td>callable</td>
<td>fflush</td>
<td>getchar</td>
<td>modff</td>
<td>sinhf</td>
<td>wcs[a-z]*</td>
</tr>
<tr>
<td>ceil</td>
<td>fgetc</td>
<td>getenv</td>
<td>modfl</td>
<td>sinhl</td>
<td>wctomb</td>
</tr>
<tr>
<td>ceilf</td>
<td>fgetpos</td>
<td>gets</td>
<td>perror</td>
<td>sinl</td>
<td>wcwidth</td>
</tr>
<tr>
<td>cell</td>
<td>fgetets</td>
<td>gmtime</td>
<td>pow</td>
<td>sprintf</td>
<td></td>
</tr>
<tr>
<td>clearerr</td>
<td>floor</td>
<td>is[a-z]*</td>
<td>powf</td>
<td>sqrt</td>
<td></td>
</tr>
<tr>
<td>clock</td>
<td>floorf</td>
<td>labs</td>
<td>powl</td>
<td>sqrtf</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The notation [a-z] indicates any lower-case letter in the portable character set. The notation * indicates any combination of digits, letters in the portable character set, and underscore.
4. The following identifiers are also reserved for use as identifiers with external linkage.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Function</th>
<th>Function</th>
<th>Function</th>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>_longjmp</td>
<td>fchmod</td>
<td>getservbyname</td>
<td>mmap</td>
<td>setitimer</td>
<td>t_error</td>
<td></td>
</tr>
<tr>
<td>_setjmp</td>
<td>fchown</td>
<td>getservbyport</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>a64l</td>
<td>fcvt</td>
<td>getserve</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>accept</td>
<td>fdtach</td>
<td>getsid</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>acosh</td>
<td>ffs</td>
<td>getsockname</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>asinh</td>
<td>fmsg</td>
<td>getssockopt</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>atan</td>
<td>fstatvfs</td>
<td>gettimeofday</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>basename</td>
<td>ftime</td>
<td>getutxent</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>bcmp</td>
<td>ftok</td>
<td>getutxid</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>bcopy</td>
<td>fnruncate</td>
<td>getutxline</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>bind</td>
<td>gcvt</td>
<td>getwd</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>brk</td>
<td>getcontext</td>
<td>granpt</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>bsd_signal</td>
<td>getdate</td>
<td>htonl</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>bzero</td>
<td>getpagesize</td>
<td>iolog</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>cbrt</td>
<td>getgrent</td>
<td>index</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>closeog</td>
<td>getgripend</td>
<td>inetsys</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>connect</td>
<td>gethostbyaddr</td>
<td>inet_addr</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>dbm_clearerr</td>
<td>gethostbyname</td>
<td>inet_nlna</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>dbm_close</td>
<td>gethostent</td>
<td>inet_makeaddr</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>dbm_delete</td>
<td>gethostdid</td>
<td>inet_netof</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>dbm_error</td>
<td>gethostname</td>
<td>inetsys</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>dbm_fetch</td>
<td>getttime</td>
<td>inetsys</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>dbm_firstkey</td>
<td>getmsg</td>
<td>instate</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>dbm_nextkey</td>
<td>getnetbyaddr</td>
<td>insq</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>dbm_open</td>
<td>getnetbyname</td>
<td>ioct</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>dbm_store</td>
<td>getnetent</td>
<td>isastream</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>dirname</td>
<td>getpagesize</td>
<td>killpg</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>ecvt</td>
<td>getpeername</td>
<td>l64a</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>endgrent</td>
<td>getpgid</td>
<td>lchown</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>endhostent</td>
<td>getpmmsg</td>
<td>listen</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>endnetent</td>
<td>getpriority</td>
<td>lockf</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>endprotoent</td>
<td>getprotobynam</td>
<td>log1p</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>endpwent</td>
<td>getprotobynum</td>
<td>logb</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>endservent</td>
<td>getprotoent</td>
<td>lstat</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>endutxent</td>
<td>getpwent</td>
<td>makecontext</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>expm1</td>
<td>getlimit</td>
<td>mknod</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>fastach</td>
<td>getusage</td>
<td>mktemp</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
<tr>
<td>fchdir</td>
<td>getservbyname</td>
<td>mktemp</td>
<td>mprotect</td>
<td>setlogmask</td>
<td>t_free</td>
<td></td>
</tr>
</tbody>
</table>

All the identifiers defined in this document that have external linkage are always reserved for use as identifiers with external linkage.

No other identifiers are reserved.

Applications must not declare or define identifiers with the same name as an identifier reserved in the same context. Since macro names are replaced whenever found, independent of scope and name space, macro names matching any of the reserved identifier names must not be defined if any associated header is included.

Except as described below, headers may be included in any order, and each may be included more than once in a given scope, with no difference in effect from that of being included only once.

- If both `<stdarg.h>` and `<varargs.h>` are included, the behaviour is unspecified.
- The effect of each inclusion of `<assert.h>` depends on the definition of NDEBUG.
- The effect of including `<regexp.h>` depends on the definitions of INIT, GETC, PEEK, UNGETC, RETURN and ERROR.
If used, a header must be included outside of any external declaration or definition, and it must be first included before the first reference to any type or macro it defines, or to any function or object it declares. However, if an identifier is declared or defined in more than one header, the second and subsequent associated headers may be included after the initial reference to the identifier. Prior to the inclusion of a header, the program must not define any macros with names lexically identical to symbols defined by that header.
2.3 Error Numbers

Most functions provide an error number in errno, which is either a variable or macro defined in <errno.h>; the macro expands to a modifiable lvalue of type int.

The value of errno is defined only after a call to a function for which it is explicitly stated to be set, and until it is changed by the next function call. The value of errno should be examined only when it is indicated to be valid by a function’s return value, or as directed in the APPLICATION USAGE sections on various entries. No function in this document sets errno to 0 to indicate an error. The value of errno may be set non-zero by a library call whether or not there is an error, provided the use of errno is not documented in the description of the function in this document.

If more than one error occurs in processing a function call, any one of the possible errors may be returned, as the order of detection is undefined.

Implementations may support additional errors not included in this list, may generate errors included in this list under circumstances other than those described here, or may contain extensions or limitations that prevent some errors from occurring. The ERRORS section on each page specifies whether an error will be returned, or whether it may be returned. Implementations will not generate a different error number from the ones described here for error conditions described in this document, but may generate additional errors unless explicitly disallowed for a particular function.

The following symbolic names identify the possible error numbers, in the context of the functions specifically defined in this document; these general descriptions are more precisely defined in the ERRORS sections of the functions that return them. Only these symbolic names should be used in programs, since the actual value of the error number is unspecified. All values listed in this section are unique, except as noted below. The values for all these names can be found in the header <errno.h>.

UX [E2BIG] Argument list too long
The sum of the number of bytes used by the new process image’s argument list and environment list is greater than the system-imposed limit of ARG_MAX bytes.

UX [EACCES] Permission denied
An attempt was made to access a file in a way forbidden by its file access permissions.

UX [EADDRINUSE] Address in use
The specified address is in use.

UX [EADDRNOTAVAIL] Address not available
The specified address is not available from the local system.

UX [EAFNOSUPPORT] Address family not supported
The implementation does not support the specified address family, or the specified address is not a valid address for the address family of the specified socket.

UX [EAGAIN] Resource temporarily unavailable
This is a temporary condition and later calls to the same routine may complete normally.

UX [EALREADY] Connection already in progress
A connection request is already in progress for the specified socket.
<table>
<thead>
<tr>
<th>Error Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBADF</td>
<td>Bad file descriptor. A file descriptor argument is out of range, refers to no open file, or a read (write) request is made to a file that is only open for writing (reading).</td>
</tr>
<tr>
<td>EBADMSG</td>
<td>Bad message. During a <code>read()</code>, <code>getmsg()</code>, or <code>ioctl</code> I_RECVFD request to a STREAMS device, a message arrived at the head of the STREAM that is inappropriate for the function receiving the message.</td>
</tr>
<tr>
<td></td>
<td>- <code>read()</code>—message waiting to be read on a STREAM is not a data message.</td>
</tr>
<tr>
<td></td>
<td>- <code>getmsg()</code>—a file descriptor was received instead of a control message.</td>
</tr>
<tr>
<td></td>
<td>- <code>ioctl()</code>—control or data information was received instead of a file descriptor when I_RECVFD was specified.</td>
</tr>
<tr>
<td>EBUSY</td>
<td>Resource busy. An attempt was made to make use of a system resource that is not currently available, as it is being used by another process in a manner that would have conflicted with the request being made by this process.</td>
</tr>
<tr>
<td>ECHILD</td>
<td>No child process. A <code>wait()</code> or <code>waitpid()</code> function was executed by a process that had no existing or unwaited-for child process.</td>
</tr>
<tr>
<td>ECONNABORTED</td>
<td>Connection aborted. The connection has been aborted.</td>
</tr>
<tr>
<td>ECONNREFUSED</td>
<td>Connection refused. An attempt to connect to a socket was refused because there was no process listening or because the queue of connection requests was full and the underlying protocol does not support retransmissions.</td>
</tr>
<tr>
<td>ECONNRESET</td>
<td>Connection reset. The connection was forcibly closed by the peer.</td>
</tr>
<tr>
<td>EDEADLK</td>
<td>Resource deadlock would occur. An attempt was made to lock a system resource that would have resulted in a deadlock situation.</td>
</tr>
<tr>
<td>EDESTADDRREQ</td>
<td>Destination address required. No bind address was established.</td>
</tr>
<tr>
<td>EDOM</td>
<td>Domain error. An input argument is outside the defined domain of the mathematical function. (Defined in the ISO C standard.)</td>
</tr>
<tr>
<td>EDQUOT</td>
<td>Reserved.</td>
</tr>
<tr>
<td>EEXIST</td>
<td>File exists. An existing file was mentioned in an inappropriate context, for instance, as a new link name in the <code>link()</code> function.</td>
</tr>
<tr>
<td>EFAULT</td>
<td>Bad address. The system detected an invalid address in attempting to use an argument of a call. The reliable detection of this error cannot be guaranteed, and when not detected may result in the generation of a signal, indicating an address...</td>
</tr>
</tbody>
</table>
violation, which is sent to the process.

**EFBIG**
File too large
The size of a file would exceed the maximum file size of an implementation.

**EHOSTUNREACH**
Host is unreachable
The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).

**EIDRM**
Identifier removed
Returned during interprocess communication if an identifier has been removed from the system.

**EILSEQ**
Illegal byte sequence
A wide-character code has been detected that does not correspond to a valid character, or a byte sequence does not form a valid wide character code.

**EINPROGRESS**
Connection in progress
O_NONBLOCK is set for the socket file descriptor and the connection cannot be immediately established.

**EINTR**
Interrupted function call
An asynchronous signal was caught by the process during the execution of an interruptible function. If the signal handler performs a normal return, the interrupted function call may return this condition. (See `<signal.h>`.)

**EINVAL**
Invalid argument
Some invalid argument was supplied; (for example, mentioning an undefined signal in a `signal()` function or a `kill()` function).

**EIO**
Input/output error
Some physical input or output error has occurred. This error may be reported on a subsequent operation on the same file descriptor. Any other error-causing operation on the same file descriptor may cause the [EIO] error indication to be lost.

**EISCONN**
Socket is connected
The specified socket is already connected.

**EISDIR**
Is a directory
An attempt was made to open a directory with write mode specified.

**ELOOP**
Too many levels of symbolic links
Too many symbolic links were encountered in resolving a pathname.

**EMFILE**
Too many open files
An attempt was made to open more than the maximum number of [OPEN_MAX] file descriptors allowed in this process.

**EMLINK**
Too many links
An attempt was made to have the link count of a single file exceed [LINK_MAX].

**EMSGSIZE**
Message too large
A message sent on a transport provider was larger than an internal message buffer or some other network limit.

**EMULTIHOP**
Reserved
<table>
<thead>
<tr>
<th>Error Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ENAMETOOLONG]</td>
<td>Filename too long. The length of a pathname exceeds {PATH_MAX}, or a pathname component is longer than {NAME_MAX} and {_POSIX_NO_TRUNC} was in effect for that file.</td>
</tr>
<tr>
<td>[ENETDOWN]</td>
<td>Network is down. The local interface used to reach the destination is down.</td>
</tr>
<tr>
<td>[ENETUNREACH]</td>
<td>Network unreachable. No route to the network is present.</td>
</tr>
<tr>
<td>[ENFILE]</td>
<td>Too many files open in system. Too many files are currently open in the system. The system has reached its predefined limit for simultaneously open files and temporarily cannot accept requests to open another one.</td>
</tr>
<tr>
<td>[ENOBUS]</td>
<td>No buffer space available. Insufficient buffer resources were available in the system to perform the socket operation.</td>
</tr>
<tr>
<td>[ENODATA]</td>
<td>No message available. No message is available on the STREAM head read queue.</td>
</tr>
<tr>
<td>[ENODEV]</td>
<td>No such device. An attempt was made to apply an inappropriate function to a device; for example, trying to read a write-only device such as a printer.</td>
</tr>
<tr>
<td>[ENOENT]</td>
<td>No such file or directory. A component of a specified pathname does not exist, or the pathname is an empty string.</td>
</tr>
<tr>
<td>[ENOEEXEC]</td>
<td>Executable file format error. A request is made to execute a file that, although it has the appropriate permissions, is not in the format required by the implementation for executable files.</td>
</tr>
<tr>
<td>[ENOLCK]</td>
<td>No locks available. A system-imposed limit on the number of simultaneous file and record locks has been reached and no more are currently available.</td>
</tr>
<tr>
<td>[ENOLINK]</td>
<td>Reserved.</td>
</tr>
<tr>
<td>[ENOMEM]</td>
<td>Not enough space. The new process image requires more memory than is allowed by the hardware or system-imposed memory management constraints.</td>
</tr>
<tr>
<td>[ENOMESG]</td>
<td>No message of the desired type. The message queue does not contain a message of the required type during interprocess communication.</td>
</tr>
<tr>
<td>[ENOPROTOOPT]</td>
<td>Protocol not available. The protocol option specified to <code>setsockopt()</code> is not supported by the implementation.</td>
</tr>
</tbody>
</table>
| [ENOSPC] | No space left on a device. During the `write()` function on a regular file or when extending a directory,
there is no free space left on the device.

UX  [ENOSR]  No STREAM resources
Insufficient STREAMS memory resources are available to perform a
STREAMS related function. This is a temporary condition; one may recover
from it if other processes release resources.

UX  [ENOSTR]  Not a STREAM
A STREAM function was attempted on a file descriptor that was not
associated with a STREAMS device.

[ENOSYS]  Function not implemented
An attempt was made to use a function that is not available in this
implementation.

UX  [ENOTCONN]  Socket not connected
The socket is not connected.

[ENOTDIR]  Not a directory
A component of the specified pathname exists, but it is not a directory, when a
directory was expected.

[ENOTEMPTY]  Directory not empty
A directory with entries other than dot and dot-dot was supplied when an
directory was expected.

UX  [ENOTSOCK]  Not a socket
The file descriptor does not refer to a socket.

[ENOTTY]  Inappropriate I/O control operation
A control function has been attempted for a file or special file for which the
operation is inappropriate.

[ENXIO]  No such device or address
Input or output on a special file refers to a device that does not exist, or makes
a request beyond the limits of the device. It may also occur when, for example,
a tape drive is not on-line.

UX  [EOPNOTSUPP]  Operation not supported on socket
The type of socket (address family or protocol) does not support the requested
operation.

UX  [EOVERFLOW]  Value too large to be stored in data type
The user ID or group ID of an IPC or file system object was too large to be
stored into appropriate member of the caller-provided structure. This error
will only occur on implementations that support a larger range of user ID or
group ID values than the declared structure member can support. This
usually occurs because the IPC or file system object resides on a remote
machine with a larger value of the type uid_t, off_t or gid_t than the local
system.

[EPERM]  Operation not permitted
An attempt was made to perform an operation limited to processes with
appropriate privileges or to the owner of a file or other resource.

[EPIPE]  Broken pipe
A write was attempted on a socket, pipe or FIFO for which there is no process
to read the data.
Error Numbers

General Information

UX  [EPROTO]  Protocol error
Some protocol error occurred. This error is device specific, but is generally not related to a hardware failure.

UX  [EPROTONOSUPPORT]  Protocol not supported
The protocol is not supported by the address family, or the protocol is not supported by the implementation.

UX  [EPROTOTYPE]  Socket type not supported
The socket type is not supported by the protocol.

[ERANGE]  Result too large or too small
The result of the function is too large (overflow) or too small (underflow) to be represented in the available space. (Defined in the ISO C standard.)

[EROFS]  Read-only file system
An attempt was made to modify a file or directory on a file system that is read only.

[ESPIPE]  Invalid seek
An attempt was made to access the file offset associated with a pipe or FIFO.

[ESRCH]  No such process
No process can be found corresponding to that specified by the given process ID.

UX  [ESTALE]  Reserved

UX  [ETIME]  STREAM ioctl() timeout
The timer set for a STREAMS ioctl() call has expired. The cause of this error is device specific and could indicate either a hardware or software failure, or a timeout value that is too short for the specific operation. The status of the ioctl() operation is indeterminate.

UX  [ETIMEDOUT]  Connection timed out
The connection to a remote machine has timed out. If the connection timed out during execution of the function that reported this error (as opposed to timing out prior to the function being called), it is unspecified whether the function has completed some or all of the documented behaviour associated with a successful completion of the function.

EX  [ETXTBSY]  Text file busy
An attempt was made to execute a pure-procedure program that is currently open for writing, or an attempt has been made to open for writing a pure-procedure program that is being executed.

UX  [EWOULDBLOCK]  Operation would block
An operation on a socket marked as non-blocking has encountered a situation such as no data available that otherwise would have caused the function to suspend execution.

An X/Open-conforming implementation may assign the same values for [EWOULDBLOCK] and [EAGAIN].

[EXDEV]  Improper link
A link to a file on another file system was attempted.
2.3.1 Additional Error Numbers

Additional implementation-dependent error numbers may be defined in `<errno.h>`.
2.4 Standard I/O Streams

A stream is associated with an external file (which may be a physical device) by opening a file, which may involve creating a new file. Creating an existing file causes its former contents to be discarded if necessary. If a file can support positioning requests, (such as a disk file, as opposed to a terminal), then a file position indicator associated with the stream is positioned at the start (byte number 0) of the file, unless the file is opened with append mode, in which case it is implementation-dependent whether the file position indicator is initially positioned at the beginning or end of the file. The file position indicator is maintained by subsequent reads, writes and positioning requests, to facilitate an orderly progression through the file. All input takes place as if bytes were read by successive calls to \texttt{fgetc()}; all output takes place as if bytes were written by successive calls to \texttt{fputc()}. When a stream is unbuffered, bytes are intended to appear from the source or at the destination as soon as possible. Otherwise bytes may be accumulated and transmitted as a block. When a stream is fully buffered, bytes are intended to be transmitted as a block when a buffer is filled. When a stream is line buffered, bytes are intended to be transmitted as a block when a newline byte is encountered. Furthermore, bytes are intended to be transmitted as a block when a buffer is filled, when input is requested on an unbuffered stream, or when input is requested on a line-buffered stream that requires the transmission of bytes. Support for these characteristics is implementation-dependent, and may be affected via \texttt{setbuf()} and \texttt{setvbuf()}.

A file may be disassociated from a controlling stream by closing the file. Output streams are flushed (any unwritten buffer contents are transmitted) before the stream is disassociated from the file. The value of a pointer to a FILE object is indeterminate after the associated file is closed (including the standard streams).

A file may be subsequently reopened, by the same or another program execution, and its contents reclaimed or modified (if it can be repositioned at its start). If the main() function returns to its original caller, or if the exit() function is called, all open files are closed (hence all output streams are flushed) before program termination. Other paths to program termination, such as calling abort(), need not close all files properly.

The address of the FILE object used to control a stream may be significant; a copy of a FILE object may not necessarily serve in place of the original.

At program startup, three streams are predefined and need not be opened explicitly: \textit{standard input} (for reading conventional input), \textit{standard output} (for writing conventional output), and \textit{standard error} (for writing diagnostic output). When opened, the standard error stream is not fully buffered; the standard input and standard output streams are fully buffered if and only if the stream can be determined not to refer to an interactive device.

2.4.1 Interaction of File Descriptors and Standard I/O Streams

An open file description may be accessed through a file descriptor, which is created using functions such as \texttt{open()} or \texttt{pipe()}, or through a stream, which is created using functions such as \texttt{fopen()} or \texttt{popen()}. Either a file descriptor or a stream will be called a handle on the open file description to which it refers; an open file description may have several handles.

Handles can be created or destroyed by explicit user action, without affecting the underlying open file description. Some of the ways to create them include \texttt{fcntl()}, \texttt{dup()}, \texttt{fdopen()}, \texttt{fileno()} and \texttt{fork()}. They can be destroyed by at least \texttt{fclose()}, \texttt{close()} and the \texttt{exec} functions.

A file descriptor that is never used in an operation that could affect the file offset (for example, \texttt{read()}, \texttt{write()} or \texttt{lseek()}) is not considered a handle for this discussion, but could give rise to one (for example, as a consequence of \texttt{fdopen()}, \texttt{dup()} or \texttt{fork()}). This exception does not include the file descriptor underlying a stream, whether created with \texttt{fopen()} or \texttt{fdopen()}, so long as it is not...
used directly by the application to affect the file offset. The `read()` and `write()` functions implicitly affect the file offset; `lseek()` explicitly affects it.

The result of function calls involving any one handle (the *active handle*) are defined elsewhere in this document, but if two or more handles are used, and any one of them is a stream, their actions must be coordinated as described below. If this is not done, the result is undefined.

A handle which is a stream is considered to be closed when either an `fclose()` or `freopen()` is executed on it (the result of `freopen()` is a new stream, which cannot be a handle on the same open file description as its previous value), or when the process owning that stream terminates with `exit()` or `abort()`. A file descriptor is closed by `close()`, `_exit()` or the `exec` functions when `FD_CLOEXEC` is set on that file descriptor.

For a handle to become the active handle, the actions below must be performed between the last use of the handle (the current active handle) and the first use of the second handle (the future active handle). The second handle then becomes the active handle. All activity by the application affecting the file offset on the first handle must be suspended until it again becomes the active file handle. (If a stream function has as an underlying function one that affects the file offset, the stream function will be considered to affect the file offset.)

The handles need not be in the same process for these rules to apply.

Note that after a `fork()`, two handles exist where one existed before. The application must assure that, if both handles will ever be accessed, that they will both be in a state where the other could become the active handle first. The application must prepare for a `fork()` exactly as if it were a change of active handle. (If the only action performed by one of the processes is one of the `exec` functions or `_exit()` (not `exit()`), the handle is never accessed in that process.)

For the first handle, the first applicable condition below applies. After the actions required below are taken, if the handle is still open, the application can close it.

- If it is a file descriptor, no action is required.
- If the only further action to be performed on any handle to this open file descriptor is to close it, no action need be taken.
- If it is a stream which is unbuffered, no action need be taken.
- If it is a stream which is line buffered, and the last byte written to the stream was a newline (that is, as if a:
  ```
  puts("\n")
  ```
  was the most recent operation on that stream), no action need be taken.
- If it is a stream which is open for writing or appending (but not also open for reading), either an `fflush()` must be done, or the stream must be closed.
- If the stream is open for reading and it is at the end of the file (`feof()` is true), no action need be taken.
- If the stream is open with a mode that allows reading and the underlying open file description refers to a device that is capable of seeking, either an `fflush()` must occur or the stream must be closed.

Otherwise, the result is undefined.
For the second handle:

- If any previous active handle has been used by a function that explicitly changed the file offset, except as required above for the first handle, the application must perform an `lseek()` or `fseek()` (as appropriate to the type of handle) to an appropriate location.

If the active handle ceases to be accessible before the requirements on the first handle, above, have been met, the state of the open file description becomes undefined. This might occur during functions such as a `fork()` or `_exit()`.

The `exec` functions make inaccessible all streams that are open at the time they are called, independent of which streams or file descriptors may be available to the new process image.

When these rules are followed, regardless of the sequence of handles used, implementations will ensure that an application, even one consisting of several processes, will yield correct results: no data will be lost or duplicated when writing, and all data will be written in order, except as requested by seeks. It is implementation-dependent whether, and under what conditions, all input is seen exactly once.

If the rules above are not followed, the result is unspecified.
2.5 STREAMS

STREAMS provides a uniform mechanism for implementing networking services and other character-based I/O. The STREAMS interface provides direct access to protocol modules. A STREAM is typically a full-duplex connection between a process and an open device or pseudo-device. However, since pipes may be STREAMS-based, a STREAM can be a full-duplex connection between two processes. The STREAM itself exists entirely within the implementation and provides a general character I/O interface for processes. It optionally includes one or more intermediate processing modules that are interposed between the process end of the STREAM (STREAM head) and a device driver at the end of the STREAM (STREAM end).

STREAMS I/O is based on messages. Messages flow in both directions in a STREAM. A given module need not understand and process every message in the STREAM, but every module in the STREAM handles every message. Each module accepts messages from one of its neighbour modules in the STREAM, and passes them to the other neighbour. For example, a line discipline module may transform the data. Data flow through the intermediate modules is bidirectional, with all modules handling, and optionally processing, all messages. There are three types of messages:

- **data messages** containing actual data for input or output
- **control data** containing instructions for the STREAMS modules and underlying implementation
- **other messages**, which include file descriptors.

The interface between the STREAM and the rest of the implementation is provided by a set of functions at the STREAM head. When a process calls `write()`, `putmsg()`, `putpmsg()` or `ioctl()`, messages are sent down the STREAM, and `read()`, `getmsg()` or `getpmsg()` accepts data from the STREAM and passes it to a process. Data intended for the device at the downstream end of the STREAM is packaged into messages and sent downstream, while data and signals from the device are composed into messages by the device driver and sent upstream to the STREAM head.

When a STREAMS-based device is opened, a STREAM is created that contains two modules: the STREAM head module and the STREAM end (driver) module. If pipes are STREAMS-based in an implementation, when a pipe is created, two STREAMS are created, each containing a STREAM head module. Other modules are added to the STREAM using `ioctl()`. New modules are "pushed" onto the STREAM one at a time in last-in, first-out (LIFO) style, as though the STREAM was a push-down stack.

**Priority**

Message types are classified according to their queueing priority and may be normal (non-priority), priority, or high-priority messages. A message belongs to a particular priority band that determines its ordering when placed on a queue. Normal messages have a priority band of 0 and are always placed at the end of the queue following all other messages in the queue. High-priority messages are always placed at the head of a queue but after any other high-priority messages already in the queue. Their priority band is ignored; they are high-priority by virtue of their type. Priority messages have a priority band greater than 0. Priority messages are always placed after any messages of the same or higher priority. High-priority and priority messages are used to send control and data information outside the normal flow of control. By convention, high-priority messages are not affected by flow control. Normal and priority messages have separate flow controls.
Message Parts

A process may access STREAMS messages that contain a data part, control part, or both. The data part is that information which is transmitted over the communication medium and the control information is used by the local STREAMS modules. The other types of messages are used between modules and are not accessible to processes. Messages containing only a data part are accessible via `putmsg()`, `putpmsg()`, `getmsg()`, `getpmsg()`, `read()` or `write()`. Messages containing a control part with or without a data part are accessible via calls to `putmsg()`, `putpmsg()`, `getmsg()` or `getpmsg()`.

2.5.1 Accessing STREAMS

A process accesses STREAMS-based files using the standard functions `open()`, `close()`, `read()`, `write()`, `ioctl()`, `pipe()`, `putmsg()`, `putpmsg()`, `getmsg()`, `getpmsg()` or `poll()`. Refer to the applicable function definitions for general properties and errors.

Calls to `ioctl()` are used to perform control functions with the STREAMS-based device associated with the file descriptor `fildes`. The arguments `command` and `arg` are passed to the STREAMS file designated by `fildes` and are interpreted by the STREAM head. Certain combinations of these arguments may be passed to a module or driver in the STREAM.

Since these STREAMS requests are a subset of `ioctl()`, they are subject to the errors described there.

STREAMS modules and drivers can detect errors, sending an error message to the STREAM head, thus causing subsequent functions to fail and set `errno` to the value specified in the message. In addition, STREAMS modules and drivers can elect to fail a particular `ioctl()` request alone by sending a negative acknowledgement message to the STREAM head. This causes just the pending `ioctl()` request to fail and set `errno` to the value specified in the message.
2.6 Interprocess Communication

The routines that provide IPC are used by a number of existing application programs. X/Open is considering the problem of generalised IPC, which is not fully addressed by the IPC interfaces provided here.

The message passing, semaphore and shared memory services form the Interprocess Communication facility. Certain aspects of their operation are common, and are described below.

2.6.1 IPC General Description

Each individual shared memory segment, message queue and semaphore set is identified by a unique positive integer, called respectively a shared memory identifier, `shmid`, a semaphore identifier, `semid`, and a message queue identifier, `msqid`. The identifiers are returned by calls on `shmget()`, `semget()` and `msgget()`, respectively.

Associated with each identifier is a data structure which contains data related to the operations which may be or may have been performed. See `<sys/shm.h>`, `<sys/sem.h>` and `<sys/msg.h>` for their descriptions.

Each of the data structures contains both ownership information and an `ipc_perm` structure, see `<sys/ipc.h>`, which are used in conjunction to determine whether or not read/write (read/alter for semaphores) permissions should be granted to processes using the IPC facilities. The `mode` member of the `ipc_perm` structure acts as a bit field which determines the permissions.

The values of the bits are given below in octal notation.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0400</td>
<td>Read by user</td>
</tr>
<tr>
<td>0200</td>
<td>Write by user</td>
</tr>
<tr>
<td>0040</td>
<td>Read by group</td>
</tr>
<tr>
<td>0020</td>
<td>Write by group</td>
</tr>
<tr>
<td>0004</td>
<td>Read by others</td>
</tr>
<tr>
<td>0002</td>
<td>Write by others</td>
</tr>
</tbody>
</table>

The name of the `ipc_perm` structure is `shm_perm`, `sem_perm` or `msg_perm`, depending on which service is being used. In each case, read and write/alter permissions are granted to a process if one or more of the following are true (xxx is replaced by `shm`, `sem` or `msg`, as appropriate):

- The process has appropriate privileges.
- The effective user ID of the process matches `xxx_perm.cuid` or `xxx_perm.uid` in the data structure associated with the IPC identifier and the appropriate bit of the `user` field in `xxx_perm.mode` is set.
- The effective user ID of the process does not match `xxx_perm.cuid` or `xxx_perm.uid` but the effective group ID of the process matches `xxx_perm.cgid` or `xxx_perm.gid` in the data structure associated with the IPC identifier, and the appropriate bit of the `group` field in `xxx_perm.mode` is set.
- The effective user ID of the process does not match `xxx_perm.cuid` or `xxx_perm.uid` and the effective group ID of the process does not match `xxx_perm.cgid` or `xxx_perm.gid` in the data structure associated with the IPC identifier, but the appropriate bit of the `other` field in `xxx_perm.mode` is set.

Otherwise, the permission is denied.
### 2.7 Data Types

All of the data types used by various system interfaces are defined by the implementation. The following table describes some of these types. Other types referenced in the description of an interface, not mentioned here, can be found in the appropriate header for that interface.

<table>
<thead>
<tr>
<th>Defined Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cc_t</td>
<td>Type used for terminal special characters.</td>
</tr>
<tr>
<td>clock_t</td>
<td>Arithmetic type used for processor times.</td>
</tr>
<tr>
<td>dev_t</td>
<td>Arithmetic type used for device numbers.</td>
</tr>
<tr>
<td>DIR</td>
<td>Type representing a directory stream.</td>
</tr>
<tr>
<td>div_t</td>
<td>Structure type returned by <code>div()</code> function.</td>
</tr>
<tr>
<td>FILE</td>
<td>A structure containing information about a file.</td>
</tr>
<tr>
<td>glob_t</td>
<td>Structure type used in pathname pattern matching.</td>
</tr>
<tr>
<td>fpos_t</td>
<td>Structure type containing all information needed to specify uniquely every position within a file.</td>
</tr>
<tr>
<td>gid_t</td>
<td>Arithmetic type used for group IDs.</td>
</tr>
<tr>
<td>icnv_t</td>
<td>Type used for conversion descriptors.</td>
</tr>
<tr>
<td>id_t</td>
<td>Arithmetic type used as a general identifier; can be used to contain at least the largest of a <code>pid_t</code>, <code>uid_t</code> or a <code>gid_t</code>.</td>
</tr>
<tr>
<td>ino_t</td>
<td>Arithmetic type used for file serial numbers.</td>
</tr>
<tr>
<td>key_t</td>
<td>Arithmetic type used for interprocess communication.</td>
</tr>
<tr>
<td>ldiv_t</td>
<td>Structure type returned by <code>ldiv()</code> function.</td>
</tr>
<tr>
<td>mode_t</td>
<td>Arithmetic type used for file attributes.</td>
</tr>
<tr>
<td>nfds_t</td>
<td>Integral type used for the number of file descriptors.</td>
</tr>
<tr>
<td>nlink_t</td>
<td>Arithmetic type used for link counts.</td>
</tr>
<tr>
<td>off_t</td>
<td>Signed Arithmetic type used for file sizes.</td>
</tr>
<tr>
<td>pid_t</td>
<td>Signed Arithmetic type used for process and process group IDs.</td>
</tr>
<tr>
<td>ptrdiff_t</td>
<td>Signed integral type of the result of subtracting two pointers.</td>
</tr>
<tr>
<td>regex_t</td>
<td>Structure type used in regular expression matching.</td>
</tr>
<tr>
<td>regmatch_t</td>
<td>Structure type used in regular expression matching.</td>
</tr>
<tr>
<td>rlim_t</td>
<td>Unsigned arithmetic type used for limit values, to which objects of type <code>int</code> and <code>off_t</code> can be cast without loss of value.</td>
</tr>
<tr>
<td>sig_atomic_t</td>
<td>Integral type of an object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts.</td>
</tr>
<tr>
<td>sigset_t</td>
<td>Integral or structure type of an object used to represent sets of signals.</td>
</tr>
<tr>
<td>size_t</td>
<td>Unsigned integral type used for size of objects.</td>
</tr>
<tr>
<td>speed_t</td>
<td>Type used for terminal baud rates.</td>
</tr>
<tr>
<td>ssize_t</td>
<td>Arithmetic type used for a count of bytes or an error indication.</td>
</tr>
<tr>
<td>tcflag_t</td>
<td>Type used for terminal modes.</td>
</tr>
<tr>
<td>time_t</td>
<td>Arithmetic type used for time in seconds.</td>
</tr>
<tr>
<td>uid_t</td>
<td>Arithmetic type used for user IDs.</td>
</tr>
<tr>
<td>useconds_t</td>
<td>Integral type used for time in microseconds.</td>
</tr>
<tr>
<td>va_list</td>
<td>Type used for traversing variable argument lists.</td>
</tr>
<tr>
<td>wchar_t</td>
<td>Integral type whose range of values can represent distinct codes for all members of the largest extended character set specified by the supported locales.</td>
</tr>
<tr>
<td>wctype_t</td>
<td>Scalar type which represents a character class descriptor.</td>
</tr>
<tr>
<td>wint_t</td>
<td>An integral type capable of storing any valid value of <code>wchar_t</code>, or <code>WEOF</code>.</td>
</tr>
<tr>
<td>wordexp_t</td>
<td>Structure type used in word expansion.</td>
</tr>
</tbody>
</table>
This chapter describes the X/Open functions, macros and external variables to support application portability at the C-language source level.
NAME
a64l, l64a — convert between 32-bit integer and radix-64 ASCII string

SYNOPSIS
UX
#include <stdlib.h>

long a64l(const char *s);

char *l64a(long value);

DESCRIPTION
These functions are used to maintain numbers stored in radix-64 ASCII characters. This is a
notation by which 32-bit integers can be represented by up to six characters; each character
represents a digit in radix-64 notation. If the type long contains more than 32 bits, only the low-
order 32 bits are used for these operations.

The characters used to represent ‘digits’ are ‘.’ for 0, ‘/’ for 1, ‘0’ through ‘9’ for 2–11,
‘A’ through ‘Z’ for 12–37, and ‘a’ through ‘z’ for 38-63.

The a64l() function takes a pointer to a radix-64 representation, in which the first digit is the
least significant, and returns a corresponding long value. If the string pointed to by s contains
more than six characters, a64l() uses the first six. If the first six characters of the string contain a
null terminator, a64l() uses only characters preceding the null terminator. The a64l() function
scans the character string from left to right with the least significant digit on the left, decoding
each character as a 6-bit radix-64 number. If the type long contains more than 32 bits, the
resulting value is sign-extended. The behaviour of a64l() is unspecified if s is a null pointer or
the string pointed to by s was not generated by a previous call to l64a().

The l64a() function takes a long argument and returns a pointer to the corresponding radix-64
representation. The behaviour of l64a() is unspecified if value is negative.

RETURN VALUE
On successful completion, a64l() returns the long value resulting from conversion of the input
string. If a string pointed to by s is an empty string, a64l() returns 0L.

The l64a() function returns a pointer to the radix-64 representation. If value is 0L, l64a() returns
a pointer to an empty string.

ERRORS
No errors are defined.

APPLICATION USAGE
The value returned by l64a() may be a pointer into a static buffer. Subsequent calls to l64a() may overwrite the buffer.

If the type long contains more than 32 bits, the result of a64l(l64a(x)) is x in the low-order 32 bits.

SEE ALSO
strtoul(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
abort — generate an abnormal process abort

SYNOPSIS
#include <stdlib.h>

void abort(void);

DESCRIPTION
The abort() function causes abnormal process termination to occur, unless the signal SIGABRT is being caught and the signal handler does not return. The abnormal termination processing includes at least the effect of fclose() on all open streams, and message catalogue descriptors, and the default actions defined for SIGABRT. The SIGABRT signal is sent to the calling process as if by means of raise() with the argument SIGABRT.

The status made available to wait() or waitpid() by abort() will be that of a process terminated by the SIGABRT signal. The abort() function will override blocking or ignoring the SIGABRT signal.

RETURN VALUE
The abort() function does not return.

ERRORS
No errors are defined.

APPLICATION USAGE
Catching the signal is intended to provide the application writer with a portable means to abort processing, free from possible interference from any implementation-provided library functions. If SIGABRT is neither caught nor ignored, and the current directory is writable, a core dump may be produced.

SEE ALSO
exit(), kill(), raise(), signal(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue for alignment with the ISO C standard and the ISO POSIX-1 standard:

- The argument list is explicitly defined as void.
- The DESCRIPTION section is revised to identify the correct order in which operations occur. It also identifies:
  — how the calling process is signalled
  — how status information is made available to the host environment
  — that abort() will override blocking or ignoring of the SIGABRT signal.

Another change is incorporated as follows:

- The APPLICATION USAGE section is replaced.
NAME
abs — return integer absolute value

SYNOPSIS
#include <stdlib.h>
int abs(int i);

DESCRIPTION
The abs() function computes the absolute value of its integer operand, i. If the result cannot be
represented, the behaviour is undefined.

RETURN VALUE
The abs() function returns the absolute value of its integer operand.

ERRORS
No errors are defined.

APPLICATION USAGE
In two's-complement representation, the absolute value of the negative integer with largest
magnitude INT_MIN might not be representable.

SEE ALSO
fabs(), labs(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:
  • In the APPLICATION USAGE section, the phrase "INT_MIN is undefined" is replaced
    with "INT_MIN might not be representable'.
access( )

DESCRIPTION
The access() function checks the file named by the pathname pointed to by the path argument for accessibility according to the bit pattern contained in amode, using the real user ID in place of the effective user ID and the real group ID in place of the effective group ID.

The value of amode is either the bitwise inclusive OR of the access permissions to be checked (R_OK, W_OK, X_OK) or the existence test, F_OK.

If any access permissions are to be checked, each will be checked individually, as described in the XBD specification, Chapter 2, Definitions. If the process has appropriate privileges, an implementation may indicate success for X_OK even if none of the execute file permission bits are set.

RETURN VALUE
If the requested access is permitted, access() succeeds and returns 0. Otherwise, –1 is returned and errno is set to indicate the error.

ERRORS
The access() function will fail if:

[EACCES]  Permission bits of the file mode do not permit the requested access, or search permission is denied on a component of the path prefix.

[EINVAL]  The value of the amode argument is invalid.

[ENAMETOOLONG]  Pathname resolution of a symbolic link produced an intermediate result whose length exceeds [PATH_MAX].

[ENOENT]  A component of path does not name an existing file or path is an empty string.

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

[EROFS]  Write access is requested for a file on a read-only file system.

[ETXTBSY]  Write access is requested for a pure procedure (shared text) file that is being executed.

APPLICATION USAGE
Additional values of amode other than the set defined in the description may be valid, for example, if a system has extended access controls.

SEE ALSO
chmod(), stat(), <unistd.h>.
CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:

- The type of argument *path* is changed from `char *` to `const char *`.

The following change is incorporated for alignment with the FIPS requirements:

- In the **ERRORS** section, the condition whereby **[ENAMETOOLONG]** will be returned if a pathname component is larger that {NAME_MAX} is now defined as mandatory and marked as an extension.

Issue 4, Version 2
The **ERRORS** section is updated for X/OPEN UNIX conformance as follows:

- It states that **[ELOOP]** will be returned if too many symbolic links are encountered during pathname resolution.

- A second **[ENAMETOOLONG]** condition is defined that may report excessive length of an intermediate result of pathname resolution of a symbolic link.
NAME
acos — arc cosine function

SYNOPSIS
#include <math.h>

double acos(double x);

DESCRIPTION
The acos() function computes the principal value of the arc cosine of x. The value of x should be in the range [−1,1].

RETURN VALUE
Upon successful completion, acos() returns the arc cosine of x, in the range [0, π] radians. If the value of x is not in the range [−1,1], and is not ±Inf or NaN, either 0.0 or NaN is returned and errno is set to [EDOM].

EX
If x is NaN, NaN is returned and errno may be set to [EDOM]. If x is ±Inf, either 0.0 is returned and errno is set to [EDOM], or NaN is returned and errno may be set to [EDOM].

ERRORS
The acos() function will fail if:

EX [EDOM] The value x is not ±Inf or NaN and is not in the range [−1,1].

The acos() function may fail if:

EX [EDOM] The value x is ±Inf or NaN.

EX No other errors will occur.

APPLICATION USAGE
An application wishing to check for error situations should set errno to 0 before calling acos(). If errno is non-zero on return, or the value NaN is returned, an error has occurred.

SEE ALSO
cos(), isnan(), <math.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

• Removed references to matherr().

• The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the ISO C standard and to rationalise error handling in the mathematics functions.

• The return value specified for [EDOM] is marked as an extension.
NAME
acosh, asinh, atanh — inverse hyperbolic functions

SYNOPSIS

```c
#include <math.h>

double acosh(double x);

double asinh(double x);

double atanh(double x);
```

DESCRIPTION
The `acosh()`, `asinh()` and `atanh()` functions compute the inverse hyperbolic cosine, sine, and tangent of their argument, respectively.

RETURN VALUE
The `acosh()`, `asinh()` and `atanh()` functions return the inverse hyperbolic cosine, sine, and tangent of their argument, respectively.

The `acosh()` function returns an implementation-dependent value (NaN or equivalent if available) and sets `errno` to [EDOM] when its argument is less than 1.0.

The `atanh()` function returns an implementation-dependent value (NaN or equivalent if available) and sets `errno` to [EDOM] when its argument has absolute value greater than 1.0.

If `x` is NaN, the `asinh()`, `acosh()` and `atanh()` functions return NaN and may set `errno` to [EDOM].

ERRORS
The `acosh()` function will fail if:

[EDOM] The `x` argument is less than 1.0.

The `atanh()` function will fail if:

[EDOM] The `x` argument has an absolute value greater than 1.0.

The `atanh()` function will fail if:

[ERANGE] The `x` argument has an absolute value equal to 1.0

The `asinh()`, `acosh()` and `atanh()` functions may fail if:

[EDOM] The value of `x` is NaN.

SEE ALSO
`cosh()`, `sinh()`, `tanh()`, `<math.h>`.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME

advance — pattern match given a compiled regular expression (TO BE WITHDRAWN)

SYNOPSIS

EX

```c
#include <regexp.h>

int advance(const char *string, const char *expbuf);
```

DESCRIPTION

Refer to regexp().

CHANGE HISTORY

First released in Issue 2.

Derived from Issue 2 of the SVID.

Issue 4

The following changes are incorporated in this issue:

- The header `<regexp.h>` is added to the SYNOPSIS section.
- The type of arguments `string` and `expbuf` are changed from `char *` to `const char *`.
- The interface is marked TO BE WITHDRAWN, because improved functionality is now provided by interfaces introduced for alignment with the ISO POSIX-2 standard.
alarm()  

NAME
alarm — schedule an alarm signal

SYNOPSIS
#include <unistd.h>

unsigned int alarm(unsigned int seconds);

DESCRIPTION
The alarm() function causes the system to send the calling process a SIGALRM signal after the number of real-time seconds specified by seconds have elapsed. Processor scheduling delays may prevent the process from handling the signal as soon as it is generated.

If seconds is 0, a pending alarm request, if any, is cancelled.

Alarm requests are not stacked; only one SIGALRM generation can be scheduled in this manner; if the SIGALRM signal has not yet been generated, the call will result in rescheduling the time at which the SIGALRM signal will be generated.

UX Interactions between alarm() and any of setitimer(), ualarm() or usleep() are unspecified.

RETURN VALUE
If there is a previous alarm() request with time remaining, alarm() returns a non-zero value that is the number of seconds until the previous request would have generated a SIGALRM signal. Otherwise, alarm() returns 0.

ERRORS
The alarm() function is always successful, and no return value is reserved to indicate an error.

APPLICATION USAGE
The fork() function clears pending alarms in the child process. A new process image created by one of the exec functions inherits the time left to an alarm signal in the old process' image.

SEE ALSO
exec, fork(), getitimer(), pause(), sigaction(), ualarm(), usleep(), <signal.h>, <unistd.h>.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:
• The header <unistd.h> is included in the SYNOPSIS section.

Issue 4, Version 2
The DESCRIPTION is updated to indicate that interactions with the setitimer(), ualarm() and usleep() functions are unspecified.
NAME
asctime — convert date and time to string

SYNOPSIS
#include <time.h>
char *asctime(const struct tm *timeptr);

DESCRIPTION
The asctime() function converts the broken-down time in the structure pointed to by timeptr into
a string in the form:
Sun Sep 16 01:03:52 1973
using the equivalent of the following algorithm:

char *asctime(const struct tm *timeptr)
{
    static char wday_name[7][3] = {
        "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
    };
    static char mon_name[12][3] = {
        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
    };
    static char result[26];
    sprintf(result, "%.3s %.3s%3d %.2d:%.2d:%.2d %d
",
            wday_name[timeptr->tm_wday],
            mon_name[timeptr->tm_mon],
            timeptr->tm_mday, timeptr->tm_hour,
            timeptr->tm_min, timeptr->tm_sec,
            1900 + timeptr->tm_year);
    return result;
}

The tm structure is defined in the header <time.h>.

RETURN VALUE
Upon successful completion, asctime() returns a pointer to the string.

ERRORS
No errors are defined.

APPLICATION USAGE
The asctime(), ctime(), gmtime() and localtime() functions return values in one of two static
objects: a broken-down time structure and an array of type char. Execution of any of the
functions may overwrite the information returned in either of these objects by any of the other
functions.

Values for the broken-down time structure can be obtained by calling gmtime() or localtime().
This interface is included for compatibility with older implementations, and does not support
localised date and time formats. Applications should use strftime() to achieve maximum
portability.

SEE ALSO
clock(), ctime(), difftime(), gmtime(), localtime(), mktime(), strftime(), strptime(), time(), utime(),
<time.h>.
CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
- The type of argument timeptr is changed from struct tm* to const struct tm*.

Other changes are incorporated as follows:
- The location of the tm structure is now defined.
- The APPLICATION USAGE section is expanded to describe the time-handling functions generally and to refer users to strftime(), which is a locale-dependent time-handling function.
NAME
asin — arc sine function

SYNOPSIS
#include <math.h>

double asin(double x);

DESCRIPTION
The asin() function computes the principal value of the arc sine of x. The value of x should be in the range [-1,1].

RETURN VALUE
Upon successful completion, asin() returns the arc sine of x, in the range [-π/2, π/2] radians. If the value of x is not in the range [-1,1], and is not ±Inf or NaN, either 0.0 or NaN is returned and errno is set to [EDOM].

EX If x is NaN, NaN is returned and errno may be set to [EDOM].

EX If x is ±Inf, either 0.0 is returned and errno is set to [EDOM] or NaN is returned and errno may be set to [EDOM].

If the result underflows, 0.0 is returned and errno may be set to [ERANGE].

ERRORS
The asin() function will fail if:

EX [EDOM] The value x is not ±Inf or NaN and is not in the range [-1,1].

The asin() function may fail if:

EX [EDOM] The value of x is ±Inf or NaN.

[ERANGE] The result underflows.

EX No other errors will occur.

APPLICATION USAGE
An application wishing to check for error situations should set errno to 0, then call asin(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

SEE ALSO
isnan(), sin(), <math.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

• Removed references to matherr().

• The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the ISO C standard and to rationalise error handling in the mathematics functions.

• The return value specified for [EDOM] is marked as an extension.
NAME
asinh — hyperbolic arc sine

SYNOPSIS
UX

#include <math.h>

double asinh(double x);

DESCRIPTION
Refer to acosh().

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
assert — insert program diagnostics

SYNOPSIS
#include <assert.h>
void assert(int expression);

DESCRIPTION
The assert() macro inserts diagnostics into programs. When it is executed, if expression is false (that is, compares equal to 0), assert() writes information about the particular call that failed (including the text of the argument, the name of the source file and the source file line number — the latter are respectively the values of the preprocessing macros __FILE__ and __LINE__) on stderr and calls abort().

Forcing a definition of the name NDEBUG, either from the compiler command line or with the preprocessor control statement #define NDEBUG ahead of the #include <assert.h> statement, will stop assertions from being compiled into the program.

RETURN VALUE
The assert() macro returns no value.

ERRORS
No errors are defined.

SEE ALSO
abort(), stderr(), <assert.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:

- The APPLICATION USAGE section is merged into the DESCRIPTION section.
NAME
atan — arc tangent function

SYNOPSIS
#include <math.h>

double atan(double x);

DESCRIPTION
The atan() function computes the principal value of the arc tangent of x.

RETURN VALUE
Upon successful completion, atan() returns the arc tangent of x in the range \([-\pi/2, \pi/2]\) radians.

EX
If x is NaN, NaN is returned and errno may be set to [EDOM].
If the result underflows, 0.0 is returned and errno may be set to [ERANGE].

ERRORS
The atan() function may fail if:

EX
[EDOM] The value of x is NaN.
[ERANGE] The result underflows.

EX
No other errors will occur.

APPLICATION USAGE
An application wishing to check for error situations should set errno to 0 before calling atan(). If
errno is non-zero on return, or the return value is NaN, an error has occurred.

SEE ALSO
atan2(), isnan(), tan(), <math.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

- Removed references to matherr().
- The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with
  the ISO C standard and to rationalise error handling in the mathematics functions.
- The return value specified for [EDOM] is marked as an extension.
NAME
atan2 — arc tangent function

SYNOPSIS
#include <math.h>

double atan2(double y, double x);

DESCRIPTION
The atan2() function computes the principal value of the arc tangent of y/x, using the signs of both arguments to determine the quadrant of the return value.

RETURN VALUE
Upon successful completion, atan2() returns the arc tangent of y/x in the range [-π, π] radians. If both arguments are 0.0, an implementation-dependent value is returned and errno may be set to [EDOM].

EX
If x or y is NaN, NaN is returned and errno may be set to [EDOM].

If the result underflows, 0.0 is returned and errno may be set to [ERANGE].

ERRORS
The atan2() function may fail if:

EX [EDOM] Both arguments are 0.0 or one or more of the arguments is NaN.
[ERANGE] The result underflows.

EX No other errors will occur.

APPLICATION USAGE
An application wishing to check for error situations should set errno to 0 before calling atan2(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

SEE ALSO
atan(), isnan(), tan(), <math.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:
• Removed references to matherr().
• The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the ISO C standard and to rationalise error handling in the mathematics functions.
• The return value specified for [EDOM] is marked as an extension.
NAME
atanh — hyperbolic arc tangent

SYNOPSIS
UX
#include <math.h>

double atanh(double x);

DESCRIPTION
Refer to acosh().

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
atexit — register function to run at process termination

SYNOPSIS
#include <stdlib.h>

int atexit(void (*func)(void));

DESCRIPTION
The atexit() function registers the function pointed to by func to be called without arguments. At normal process termination, functions registered by atexit() are called in the reverse order to that in which they were registered. Normal termination occurs either by a call to exit() or a return from main().

At least 32 functions can be registered with atexit().

After a successful call to any of the exec functions, any functions previously registered by atexit() are no longer registered.

RETURN VALUE
Upon successful completion, atexit() returns 0. Otherwise, it returns a non-zero value.

ERRORS
No errors are defined.

APPLICATION USAGE
The functions registered by a call to atexit() must return to ensure that all registered functions are called.

UX The application should call sysconf() to obtain the value of {ATEXIT_MAX}, the number of functions that can be registered. There is no way for an application to tell how many functions have already been registered with atexit().

SEE ALSO
exit(), sysconf(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the ANSI C standard.

Issue 4, Version 2
The APPLICATION USAGE section is updated to indicate how an application can determine the setting of {ATEXIT_MAX}, which is a constant added for X/OPEN UNIX conformance.
NAME

atof — convert string to double-precision number

SYNOPSIS

#include <stdlib.h>

double atof(const char * str);

DESCRIPTION

The call atof(str) is equivalent to:

    strtod(str, (char **)NULL),

except that the handling of errors may differ. If the value cannot be represented, the behaviour
is undefined.

RETURN VALUE

The atof() function returns the converted value if the value can be represented.

ERRORS

No errors are defined.

APPLICATION USAGE

The atof() function is subsumed by strtod() but is retained because it is used extensively in
existing code. If the number is not known to be in range, strtod() should be used because atof() is
not required to perform any error checking.

SEE ALSO

    strtod(), <stdlib.h>.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following change is incorporated for alignment with the ISO C standard:

• The type of argument str is changed from char * to const char *.

Other changes are incorporated as follows:

• Reference to how str is converted is removed from the DESCRIPTION section.

• The APPLICATION USAGE section is added.
NAME
atoi — convert string to integer

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

DESCRIPTION
The call `atoi(str)` is equivalent to:

```
(int) strtol(str, (char **)NULL, 10)
```
except that the handling of errors may differ. If the value cannot be represented, the behaviour is undefined.

RETURN VALUE
The `atoi()` function returns the converted value if the value can be represented.

ERRORS
No errors are defined.

APPLICATION USAGE
The `atoi()` function is subsumed by `strtol()` but is retained because it is used extensively in existing code. If the number is not known to be in range, `strtol()` should be used because `atoi()` is not required to perform any error checking.

SEE ALSO
`strtol()`, `<stdlib.h>`.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:

- The type of argument `str` is changed from `char *` to `const char *`.

Other changes are incorporated as follows:

- Reference to how `str` is converted is removed from the DESCRIPTION section.
- The APPLICATION USAGE section is added.
NAME
  atol — convert string to long integer

SYNOPSIS
  #include <stdlib.h>
  long int atol(const char * str);

DESCRIPTION
  The call atol(str) is equivalent to:
       strtol(str, (char **)NULL, 10)
  except that the handling of errors may differ. If the value cannot be represented, the behaviour
  is undefined.

RETURN VALUE
  The atol() function returns the converted value if the value can be represented.

ERRORS
  No errors are defined.

APPLICATION USAGE
  The atol() function is subsumed by strtol() but is retained because it is used extensively in
  existing code. If the number is not known to be in range, strtol() should be used because atol() is
  not required to perform any error checking.

SEE ALSO
  strtol(), <stdlib.h>.

CHANGE HISTORY
  First released in Issue 1.
  Derived from Issue 1 of the SVID.

Issue 4
  The following changes are incorporated for alignment with the ISO C standard:
  • The type of argument str is changed from char * to const char *.
  • The return type of the function is expanded to long int.

Other changes are incorporated as follows:
  • Reference to how str is converted is removed from the DESCRIPTION section.
  • The APPLICATION USAGE section is added.
NAME
basename — return the last component of a pathname

SYNOPSIS
#include <libgen.h>
char *basename(char *path);

DESCRIPTION
The basename() function takes the pathname pointed to by path and returns a pointer to the final component of the pathname, deleting any trailing ‘/’ characters.
If the string consists entirely of the ‘/’ character, basename() returns a pointer to the string “/”.
If path is a null pointer or points to an empty string, basename() returns a pointer to the string “.”.

RETURN VALUE
The basename() function returns a pointer to the final component of path.

EXAMPLES

<table>
<thead>
<tr>
<th>Input String</th>
<th>Output String</th>
</tr>
</thead>
<tbody>
<tr>
<td>”/usr/lib”</td>
<td>”lib”</td>
</tr>
<tr>
<td>”/usr/”</td>
<td>”usr”</td>
</tr>
<tr>
<td>”/”</td>
<td>”/”</td>
</tr>
</tbody>
</table>

APPLICATION USAGE
The basename() function may modify the string pointed to by path, and may return a pointer to static storage that may then be overwritten by a subsequent call to basename().

ERRORS
No errors are defined.

SEE ALSO
dirname(), <libgen.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
  bcmp — memory operations

SYNOPSIS
  #include <strings.h>
  int bcmp(const void *s1, const void *s2, size_t n);

DESCRIPTION
  The bcmp() function compares the first n bytes of the area pointed to
  by s1 with the area pointed to by s2.

RETURN VALUE
  The bcmp() function returns 0 if s1 and s2 are identical, non-zero
  otherwise. Both areas are assumed to be n bytes long. If the value of
  n is 0, bcmp() returns 0.

ERRORS
  No errors are defined.

APPLICATION USAGE
  For portability to implementations conforming to earlier versions
  of this document, memcmp() is preferred over this function.

SEE ALSO
  memcmp(), <strings.h>.

CHANGE HISTORY
  First released in Issue 4, Version 2.
NAME
bcopy — memory operations

SYNOPSIS
UX
#include <strings.h>

void bcopy(const void * s1, void * s2, size_t n);

DESCRIPTION
The bcopy() function copies n bytes from the area pointed to by s1 to the area pointed to by s2.

RETURN VALUE
The bcopy() function returns no value.

ERRORS
No errors are defined.

APPLICATION USAGE
For portability to implementations conforming to earlier versions of this document, memmove() is preferred over this function.

The following are approximately equivalent (note the order of the arguments):

bcopy(s1, s2, n) ≈ memmove(s2, s1, n)

SEE ALSO
memmove(), <strings.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
brk, sbrk — change space allocation

SYNOPSIS
#include <unistd.h>

int brk(void * addr);
void *sbrk(int incr);

DESCRIPTION
The brk() and sbrk() functions are used to change the amount of space allocated for the calling
process. The change is made by resetting the process’ break value and allocating the appropriate
amount of space. The amount of allocated space increases as the break value increases. The
newly-allocated space is set to 0. However, if the application first decrements and then
increments the break value, the contents of the reallocated space are unspecified.

The brk() function sets the break value to addr and changes the allocated space accordingly.

The sbrk() function adds incr bytes to the break value and changes the allocated space
accordingly. If incr is negative, the amount of allocated space is decreased by incr bytes. The
current value of the program break is returned by sbrk(0).

The behaviour of brk() and sbrk() is unspecified if an application also uses any other memory
functions (such as malloc(), mmap(), free()). Other functions may use these other memory
functions silently.

RETURN VALUE
Upon successful completion, brk() returns 0. Otherwise, it returns −1 and sets errno to indicate
the error.

Upon successful completion, sbrk() returns the prior break value. Otherwise, it returns
(void *)−1 and sets errno to indicate the error.

ERRORS
The brk() and sbrk() functions will fail if:

[ENOMEM] The requested change would allocate more space than allowed.

The brk() and sbrk() functions may fail if:

[EAGAIN] The total amount of system memory available for allocation to this process is
temporarily insufficient. This may occur even though the space requested was
less than the maximum data segment size.

[ENOMEM] The requested change would be impossible as there is insufficient swap space
available, or would cause a memory allocation conflict.

APPLICATION USAGE
The brk() and sbrk() functions have been used in specialised cases where no other memory
allocation function provided the same capability. The use of mmap() is now preferred because it
can be used portably with all other memory allocation functions and with any function that uses
other allocation functions.

It is unspecified whether the pointer returned by sbrk() is aligned suitably for any purpose.

SEE ALSO
exec, malloc(), mmap(), <unistd.h>.
CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
bsd_signal — simplified signal facilities

SYNOPSIS
UX
#include <signal.h>

void (*bsd_signal(int sig, void (*func)(int)))(int);

DESCRIPTION
The bsd_signal() function provides a partially compatible interface for programs written to
historical system interfaces (see APPLICATION USAGE below).

The function call bsd_signal(sig, func) has an effect as if implemented as:

void (*bsd_signal(int sig, void (*func)(int)))(int)
{
    struct sigaction act, oact;
    act.sa_handler = func;
    act.sa_flags = SA_RESTART;
    sigemptyset(&act.sa_mask);
    sigaddset(&act.sa_mask, sig);
    if (sigaction(sig, &act, &oact) == -1)
        return(SIG_ERR);
    return(oact.sa_handler);
}

The handler function should be declared:

void handler(int sig);

where sig is the signal number. The behaviour is undefined if func is a function that takes more
than one argument, or an argument of a different type.

RETURN VALUE
Upon successful completion, bsd_signal() returns the previous action for sig. Otherwise,
SIG_ERR is returned and errno is set to indicate the error.

ERRORS
Refer to sigaction().

APPLICATION USAGE
This function is a direct replacement for the BSD signal() function for simple applications that
are installing a single-argument signal handler function. If a BSD signal handler function is
being installed that expects more than one argument, the application has to be modified to use
sigaction(). The bsd_signal() function differs from signal() in that the SA_RESTART flag is set
and the SA_RESETHAND will be clear when bsd_signal() is used. The state of these flags is not
specified for signal().

SEE ALSO
    sigaction(), sigaddset(), sigemptyset(), signal(), <signal.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME

bsearch — binary search a sorted table

SYNOPSIS

#include <stdlib.h>

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

DESCRIPTION

The bsearch() function searches an array of nel objects, the initial element of which is pointed to by base, for an element that matches the object pointed to by key. The size of each element in the array is specified by width.

The comparison function pointed to by compar is called with two arguments that point to the key object and to an array element, in that order.

The function must return an integer less than, equal to, or greater than 0 if the key object is considered, respectively, to be less than, to match, or to be greater than the array element. The array must consist of: all the elements that compare less than, all the elements that compare equal to, and all the elements that compare greater than the key object, in that order.

RETURN VALUE

The bsearch() function returns a pointer to a matching member of the array, or a null pointer if no match is found. If two or more members compare equal, which member is returned is unspecified.

ERRORS

No errors are defined.

EXAMPLES

The example below searches a table containing pointers to nodes consisting of a string and its length. The table is ordered alphabetically on the string in the node pointed to by each entry.

The code fragment below reads in strings and either finds the corresponding node and prints out the string and its length, or prints an error message.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define TABSIZE 1000

struct node { /* these are stored in the table */
    char *string;
    int length;
};

struct node table[TABSIZE]; /* table to be searched */

struct node *node_ptr, node;
/* routine to compare 2 nodes */
int node_compare(const void *, const void *);
char str_space[20]; /* space to read string into */

node.string = str_space;
while (scanf("%s", node.string) != EOF) {
    node_ptr = (struct node *)bsearch((void *)&node,
        (void *)table, TABSIZE,
        sizeof(struct node), node_compare);
    if (node_ptr != NULL) {
        (void)printf("string = %20s, length = %d\n",
            node_ptr->string, node_ptr->length);
    } else {
        (void)printf("not found: %s\n", node.string);
    }
}

/*
   This routine compares two nodes based on an alphabetical ordering of the string field.
*/
int node_compare(const void *node1, const void *node2)
{
    return strcoll(((const struct node *)node1)->string,
        ((const struct node *)node2)->string);
}
APPLICATION USAGE

The pointers to the key and the element at the base of the table should be of type pointer-to-element.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

In practice, the array is usually sorted according to the comparison function.

SEE ALSO

hsearch(), lsearch(), qsort(), tsearch(), <stdlib.h>.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated for alignment with the ISO C standard:

• The type of arguments key and base, and the type of arguments to compar(), are changed from void* to const void*.

• The requirement that the table be sorted according to compar() is removed from the DESCRIPTION section.

Other changes are incorporated as follows:

• Text indicating the need for various casts is removed from the APPLICATION USAGE section.

• The code in the EXAMPLES section is changed to use strcoll() instead of strcmp() in node_compare().

• The return value and the contents of the array are now requirements on the application.

• The DESCRIPTION is changed to specify the order of arguments.
NAME
bzero — memory operations

SYNOPSIS

```
#include <strings.h>

void bzero(void *, size_t);
```

DESCRIPTION
The bzero() function places n zero-valued bytes in the area pointed to by s.

RETURN VALUE
The bzero() function returns no value.

ERRORS
No errors are defined.

APPLICATION USAGE
For portability to implementations conforming to earlier versions of this document, memset() is preferred over this function.

SEE ALSO
memset(), <strings.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
calloc — memory allocator

SYNOPSIS
#include <stdlib.h>
void *calloc(size_t nelem, size_t elsize);

DESCRIPTION
The calloc() function allocates unused space for an array of nelem elements each of whose size in bytes is elsize. The space is initialised to all bits 0.

The order and contiguity of storage allocated by successive calls to calloc() is unspecified. The pointer returned if the allocation succeeds is suitably aligned so that it may be assigned to a pointer to any type of object and then used to access such an object or an array of such objects in the space allocated (until the space is explicitly freed or reallocated). Each such allocation will yield a pointer to an object disjoint from any other object. The pointer returned points to the start (lowest byte address) of the allocated space. If the space cannot be allocated, a null pointer is returned. If the size of the space requested is 0, the behaviour is implementation-dependent; the value returned will be either a null pointer or a unique pointer.

RETURN VALUE
Upon successful completion with both nelem and elsize non-zero, calloc() returns a pointer to the allocated space. If either nelem or elsize is 0, then either a null pointer or a unique pointer value that can be successfully passed to free() is returned. Otherwise, it returns a null pointer and sets errno to indicate the error.

ERRORS
The calloc() function will fail if:

EX [ENOMEM] Insufficient memory is available.

APPLICATION USAGE
There is now no requirement for the implementation to support the inclusion of <malloc.h>.

SEE ALSO
free(), malloc(), realloc(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue for alignment with the ISO C standard:

• The DESCRIPTION is updated to indicate (a) that the order and contiguity of storage allocated by successive calls to this function is unspecified, (b) that each allocation yields a pointer to an object disjoint from any other object, (c) that the returned pointer points to the lowest byte address of the allocation, and (d) the behaviour if space is requested of zero size.

• The RETURN VALUE section is updated to indicate what will be returned if either nelem or elsize is 0.

Other changes are incorporated as follows:

• The setting of errno and the [ENOMEM] error are marked as extensions.

• The APPLICATION USAGE section is changed to record that <malloc.h> need no longer be supported on XSI-conformant systems.
NAME
catclose — close a message catalogue descriptor

SYNOPSIS
EX
#include <nl_types.h>

int catclose(nl_catd catd);

DESCRIPTION
The catclose() function closes the message catalogue identified by catd. If a file descriptor is used to implement the type nl_catd, that file descriptor will be closed.

RETURN VALUE
Upon successful completion, catclose() returns 0. Otherwise −1 is returned, and errno is set to indicate the error.

ERRORS
The catclose() function may fail if:
[EBADF] The catalogue descriptor is not valid.
[EINTR] The catclose() function was interrupted by a signal.

SEE ALSO
catgets(), catopen(), <nl_types.h>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The following change is incorporated in this issue:
• The [EBADF] and [EINTR] errors are added to the ERRORS section.
NAME
    catgets — read a program message

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

DESCRIPTION
    The catgets() function attempts to read message msg_id, in set set_id, from the message catalogue
    identified by catd. The catd argument is a message catalogue descriptor returned from an earlier
    call to catopen(). The s argument points to a default message string which will be returned by
    catgets() if it cannot retrieve the identified message.

RETURN VALUE
    If the identified message is retrieved successfully, catgets() returns a pointer to an internal buffer
    area containing the null-terminated message string. If the call is unsuccessful for any reason, s is
    returned and errno may be set to indicate the error.

ERRORS
    The catgets() function may fail if:
    [EBADF] The catd argument is not a valid message catalogue descriptor open for
            reading.
    [EINTR] The read operation was terminated due to the receipt of a signal, and no data
            was transferred.
    [EINVAL] The message catalog identified by catd is corrupted.
    [ENOMSG] The message identified by set_id and msg_id is not in the message catalog.

SEE ALSO
    catclose(), catopen(), <nl_types.h>.

CHANGE HISTORY
    First released in Issue 2.

Issue 4
    The following changes are incorporated in this issue:
    • The type of argument s is changed from char * to const char *.
    • The [EBADF] and [EINTR] errors are added to the ERRORS section.

Issue 4, Version 2
    The following changes are incorporated for X/OPEN UNIX conformance:
    • The RETURN VALUE section notes that errno may be set to indicate an error.
    • In the ERRORS section, [EINVAL] and [ENOMSG] are added as optional errors.
NAME
catopen — open a message catalogue

SYNOPSIS
EX
#include <nl_types.h>

nl_catd catopen(const char *name, int oflag);

DESCRIPTION
The catopen() function opens a message catalogue and returns a message catalogue descriptor. The name argument specifies the name of the message catalogue to be opened. If name contains a “/”, then name specifies a complete name for the message catalogue. Otherwise, the environment variable NLSPATH is used with name substituted for %N (see the XBD specification, Chapter 6, Environment Variables). If NLSPATH does not exist in the environment, or if a message catalogue cannot be found in any of the components specified by NLSPATH, then an implementation-dependent default path is used. This default may be affected by the setting of LC_MESSAGES if the value of oflag is NL_CAT_LOCALE, or the LANG environment variable if oflag is 0.

A message catalogue descriptor remains valid in a process until that process closes it, or a successful call to one of the exec functions. A change in the setting of the LC_MESSAGES category may invalidate existing open catalogues.

If a file descriptor is used to implement message catalogue descriptors, the FD_CLOEXEC flag will be set; see <fcntl.h>.

If the value of the oflag argument is 0, the LANG environment variable is used to locate the catalogue without regard to the LC-messages category. If the oflag argument is NL_CAT_LOCALE, the LC_MESSAGES category is used to locate the message catalogue (see the XBD specification, Section 6.2, Internationalisation Variables).

RETURN VALUE
Upon successful completion, catopen() returns a message catalogue descriptor for use on subsequent calls to catgets() and catclose(). Otherwise catopen() returns (nl_catd) −1 and sets errno to indicate the error.

ERRORS
The catopen() function may fail if:

[EACCES] Search permission is denied for the component of the path prefix of the message catalogue or read permission is denied for the message catalogue.

[EMFILE] [OPEN_MAX] file descriptors are currently open in the calling process.

[ENAMETOOLONG] The length of the pathname of the message catalogue exceeds [PATH_MAX], or a pathname component is longer than [NAME_MAX].

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds [PATH_MAX].

[ENOENT] The message catalogue does not exist or the name argument points to an empty string.

[ENOMEM] Insufficient storage space is available.
[ENOTDIR] A component of the path prefix of the message catalogue is not a directory.

APPLICATION USAGE
Some implementations of `catopen()` use `malloc()` to allocate space for internal buffer areas. The `catopen()` function may fail if there is insufficient storage space available to accommodate these buffers.

Portable applications must assume that message catalogue descriptors are not valid after a call to one of the `exec` functions.

Application writers should be aware that guidelines for the location of message catalogues have not yet been developed. Therefore they should take care to avoid conflicting with catalogues used by other applications and the standard utilities.

SEE ALSO
`catclose()`, `catgets()`, `<fcntl.h>`, `<nl_types.h>`.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The following changes are incorporated in this issue:

- The type of argument `name` is changed from `char *` to `const char *`.
- The DESCRIPTION section is updated (a) to indicate the longevity of message catalogue descriptors, and (b) to specify values for the `oflag` argument and the effect of `LC_MESSAGES` and `NLSPATH`.
- The [EACCES], [EMFILE], [ENAMETOOLONG], [ENFILE], [ENOENT] and [ENOTDIR] errors are added to the ERRORS section.
- The APPLICATION USAGE section is updated to indicate that (a) portable applications should not assume the continued validity of message catalogue descriptors after a call to one of the `exec` functions, and (b) message catalogues must be located with care.

Issue 4, Version 2
The following change is incorporated for X/OPEN UNIX conformance:

- In the ERRORS section, an [ENAMETOOLONG] condition is defined that may report excessive length of an intermediate result of pathname resolution of a symbolic link.
NAME
cbrt — cube root function

SYNOPSIS

UX

#include <math.h>

double cbrt(double x);

DESCRIPTION

The cbrt() function computes the cube root of x.

RETURN VALUE

On successful completion, cbrt() returns the cube root of x. If x is NaN, cbrt() returns NaN and errno may be set to [EDOM].

ERRORS

The cbrt() function may fail if:

[EDOM] The value of x is NaN.

SEE ALSO

<math.h>.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
ceil — ceiling value function

SYNOPSIS
#include <math.h>

double ceil(double x);

DESCRIPTION
The ceil() function computes the smallest integral value not less than x.

RETURN VALUE
Upon successful completion, ceil() returns the smallest integral value not less than x, expressed as a type double.

EX
If x is NaN, NaN is returned and errno may be set to [EDOM].

EX
If the correct value would cause overflow, HUGE_VAL is returned and errno is set to [ERANGE]. If x is ±Inf or ±0, the value of x is returned.

ERRORS
The ceil() function will fail if:
[ERANGE] The result overflows.
The ceil() function may fail if:

EX [EDOM] The value of x is NaN.

EX No other errors will occur.

APPLICATION USAGE
The integral value returned by ceil() as a double may not be expressible as an int or long int. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.
An application wishing to check for error situations should set errno to 0 before calling ceil(). If errno is non-zero on return, or the return value is NaN, an error has occurred.
The ceil() function can only overflow when the floating point representation has DBL_MANT_DIG > DBL_MAX_EXP.

SEE ALSO
floor(), isnan(), <math.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:
• Removed references to matherr().
• The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the ISO C standard and to rationalise error handling in the mathematics functions.
• The return value specified for [EDOM] is marked as an extension.
• Support for x being ±Inf or ±0 is added to the RETURN VALUE section and marked as an extension.
NAME
cfgetispeed — get input baud rate

SYNOPSIS
#include <termios.h>
speed_t cfgetispeed(const struct termios *termios_p);

DESCRIPTION
The cfgetispeed() function extracts the input baud rate from the termios structure to which the termios_p argument points.

This function returns exactly the value in the termios data structure, without interpretation.

RETURN VALUE
Upon successful completion, cfgetispeed() returns a value of type speed_t representing the input baud rate.

ERRORS
No errors are defined.

SEE ALSO
cfgetospeed(), cfsetispeed(), cfsetospeed(), tcgetattr(), <termios.h>, the XBD specification, Chapter 9, General Terminal Interface.

CHANGE HISTORY
First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

Issue 4
The following changes are incorporated for alignment with the ISO POSIX-1 standard:

• The type of the argument termios_p is changed from struct termios* to const struct termios*.

• The DESCRIPTION section is changed to indicate that the function simply returns the value from termios_p, irrespective of how that structure was obtained. Issue 3 states that if termios_p was not obtained by a successful call to tcgetattr(), the behaviour is undefined.
NAME
cfgetospeed — get output baud rate

SYNOPSIS
#include <termios.h>
speed_t cfgetospeed(const struct termios *termios_p);

DESCRIPTION
The cfgetospeed() function extracts the output baud rate from the termios structure to which the
termios_p argument points.

This function returns exactly the value in the termios data structure, without interpretation.

RETURN VALUE
Upon successful completion, cfgetospeed() returns a value of type speed_t representing the
output baud rate.

ERRORS
No errors are defined.

SEE ALSO
cfgetispeed(), cfsetispeed(), cfsetospeed(), tcgetattr(), <termios.h>, the XBD specification, Chapter
9, General Terminal Interface.

CHANGE HISTORY
First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

Issue 4
The following changes are incorporated for alignment with the ISO POSIX-1 standard:

• The type of the argument termios_p is changed from struct termios* to const struct termios*.

• The DESCRIPTION section is changed to indicate that the function simply returns the value
from termios_p, irrespective of how that structure was obtained. Issue 3 states that if
termios_p was not obtained by a successful call to tcgetattr(), the behaviour is undefined.
NAME
cfsetispeed — set input baud rate

SYNOPSIS
#include <termios.h>
int cfsetispeed(struct termios *termios_p, speed_t speed);

DESCRIPTION
The cfsetispeed() function sets the input baud rate stored in the structure pointed to by termios_p
to speed.

There is no effect on the baud rates set in the hardware until a subsequent successful call to
tcsetattr() on the same termios structure.

RETURN VALUE
EX Upon successful completion, cfsetispeed() returns 0. Otherwise -1 is returned, and errno may be
set to indicate the error.

ERRORS
The cfsetispeed() function may fail if:
EX [EINVAL] The speed value is not a valid baud rate.
UX [EINVAL] The value of speed is outside the range of possible speed values as specified in
<termios.h>.

SEE ALSO
cfgetispeed(), cfgetospeed(), cfsetospeed(), tcsetattr(), <termios.h>, the XBD specification, Chapter
9, General Terminal Interface.

CHANGE HISTORY
First released in Issue 3.
Entry included for alignment with the POSIX.1-1988 standard.

Issue 4
The following change is incorporated in this issue:
• The first description of the [EINVAL] error is added and is marked as an extension.

Issue 4, Version 2
The ERRORS section is changed to indicate that [EINVAL] may be returned if the specified
speed is outside the range of possible speed values given in <termios.h>.
NAME
  cfsetospeed — set output baud rate

SYNOPSIS
#include <termios.h>

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

DESCRIPTION
The cfsetospeed() function sets the output baud rate stored in the structure pointed to by termios_p to speed.

There is no effect on the baud rates set in the hardware until a subsequent successful call to tcsetattr() on the same termios structure.

RETURN VALUE
Upon successful completion, cfsetospeed() returns 0. Otherwise it returns −1 and errno may be set to indicate the error.

ERRORS
The cfsetospeed() function may fail if:

EX [EINVAL] The speed value is not a valid baud rate.

UX [EINVAL] The value of speed is outside the range of possible speed values as specified in <termios.h>.

SEE ALSO
cfgetispeed(), cfgetospeed(), cfsetispeed(), tcsetattr(), <termios.h>, the XBD specification, Chapter 9, General Terminal Interface.

CHANGE HISTORY
First released in Issue 3.
Entry included for alignment with the POSIX.1-1988 standard.

Issue 4
The following change is incorporated in this issue:
  • The first description of the [EINVAL] error is added and is marked as an extension.

Issue 4, Version 2
The ERRORS section is changed to indicate that [EINVAL] may be returned if the specified speed is outside the range of possible speed values given in <termios.h>.
NAME
  chdir — change working directory

SYNOPSIS
  #include <unistd.h>
  int chdir(const char *path);

DESCRIPTION
  The chdir() function causes the directory named by the pathname pointed to by the path argument to become the current working directory; that is, the starting point for path searches for pathnames not beginning with `/.

RETURN VALUE
  Upon successful completion, 0 is returned. Otherwise, −1 is returned, the current working directory remains unchanged and errno is set to indicate the error.

ERRORS
  The chdir() function will fail if:
  [EACCES] Search permission is denied for any component of the pathname.
  UX [ELOOP] Too many symbolic links were encountered in resolving path.
  FIPS [ENAMETOOLONG] The path argument exceeds [PATH_MAX] in length or a pathname component is longer than [NAME_MAX].
  [ENOENT] A component of path does not name an existing directory or path is an empty string.
  [ENOTDIR] A component of the pathname is not a directory.

The chdir() function may fail if:
  UX [ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds [PATH_MAX].

SEE ALSO
  chroot(), getcwd(), <unistd.h>.

CHANGE HISTORY
  First released in Issue 1.
  Derived from Issue 1 of the SVID.

Issue 4
  The following change is incorporated for alignment with the ISO POSIX-1 standard:
      • The type of argument path is changed from char * to const char *.

  The following change is incorporated for alignment with the FIPS requirements:
      • In the ERRORS section, the condition whereby [ENAMETOOLONG] will be returned if a pathname component is larger than [NAME_MAX] is now defined as mandatory and marked as an extension.

  Another change is incorporated as follows:
      • The header <unistd.h> is added to the SYNOPSIS section.
Issue 4, Version 2

The ERRORS section is updated for X/OPEN UNIX conformance as follows:

- It states that [ELOOP] will be returned if too many symbolic links are encountered during pathname resolution.
- A second [ENAMETOOLONG] condition is defined that may report excessive length of an intermediate result of pathname resolution of a symbolic link.
NAME
chmod — change mode of file

SYNOPSIS
#include <sys/types.h>
#include <sys/stat.h>

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

DESCRIPTION
The chmod() function changes S_ISUID, S_ISGID, S_ISVTX and the file permission bits of the file named by the pathname pointed to by the path argument to the corresponding bits in the mode argument. The effective user ID of the process must match the owner of the file or the process must have appropriate privileges in order to do this.

S_ISUID, S_ISGID and the file permission bits are described in <sys/stat.h>.

If a directory is writable and the mode bit S_ISVTX is set on the directory, a process may remove or rename files within that directory only if one or more of the following is true:

• The effective user ID of the process is the same as that of the owner ID of the file.
• The effective user ID of the process is the same as that of the owner ID of the directory.
• The process has appropriate privileges.

If the calling process does not have appropriate privileges, and if the group ID of the file does not match the effective group ID or one of the supplementary group IDs and if the file is a regular file, bit S_ISGID (set-group-ID on execution) in the file’s mode will be cleared upon successful return from chmod().

Additional implementation-dependent restrictions may cause the S_ISUID and S_ISGID bits in mode to be ignored.

The effect on file descriptors for files open at the time of a call to chmod() is implementation-dependent.

Upon successful completion, chmod() will mark for update the st_ctime field of the file.

RETURN VALUE
Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to indicate the error. If −1 is returned, no change to the file mode will occur.

ERRORS
The chmod() function will fail if:

[EACCES] Search permission is denied on a component of the path prefix.

[ELOOP] Too many symbolic links were encountered in resolving path.

[ENAMETOOLONG] The length of the path argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

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

[ENOENT] A component of path does not name an existing file or path is an empty string.

[EPERM] The effective user ID does not match the owner of the file and the process does not have appropriate privileges.

[EROFS] The named file resides on a read-only file system.
The `chmod()` function may fail if:

- **[EINTR]** A signal was caught during execution of the function.
- **[EINVAL]** The value of the `mode` argument is invalid.
- **[ENAMETOOLONG]** Pathname resolution of a symbolic link produced an intermediate result whose length exceeds `{PATH_MAX}`.

**APPLICATION USAGE**

In order to ensure that the S_ISUID and S_ISGID bits are set, an application requiring this should use `stat()` after a successful `chmod()` in order to verify this.

Any file descriptors currently open by any process on the file may become invalid if the mode of the file is changed to a value which would deny access to that process. One situation where this could occur is on a stateless file system.

**SEE ALSO**

`chown()`, `mkdir()`, `mkfifo()`, `open()`, `stat()`, `statvfs()`, `<sys/stat.h>`, `<sys/types.h>`.

**CHANGE HISTORY**

First released in Issue 1.

Derived from Issue 1 of the SVID.

**Issue 4**

The following change is incorporated for alignment with the ISO POSIX-1 standard:

- The type of argument `path` is changed from `char *` to `const char *`.

The following change is incorporated for alignment with the FIPS requirements:

- In the `ERRORS` section, the condition whereby `[ENAMETOOLONG]` will be returned if a pathname component is larger that `{NAME_MAX}` is now defined as mandatory and marked as an extension.

Other changes are incorporated as follows:

- The header `<sys/types.h>` is now marked as optional (OH); this header need not be included on XSI-conformant systems.
- The `[EINVAL]` error is marked as an extension.

**Issue 4, Version 2**

The following changes are incorporated for X/OPEN UNIX conformance:

- The `DESCRIPTION` is updated to describe X/OPEN UNIX functionality relating to permission checks applied when removing or renaming files in a directory having the S_ISVTX bit set.
- In the `ERRORS` section, the condition whereby `[ELOOP]` will be returned if too many symbolic links are encountered during pathname resolution is defined as mandatory, and `[EINTR]` is added as an optional error.
- In the `ERRORS` section, a second `[ENAMETOOLONG]` condition is defined that may report excessive length of an intermediate result of pathname resolution of a symbolic link.
NAME
chown — change owner and group of a file

SYNOPSIS
#include <sys/types.h>
#include <unistd.h>

int chown(const char * path, uid_t owner, gid_t group);

DESCRIPTION
The path argument points to a pathname naming a file. The user ID and group ID of the named
file are set to the numeric values contained in owner and group respectively.

FIPS On XSI-conformant systems [_POSIX_CHOWN_RESTRICTED] is always defined, therefore:
• Changing the user ID is restricted to processes with appropriate privileges.
• Changing the group ID is permitted to a process with an effective user ID equal to the user
ID of the file, but without appropriate privileges, if and only if owner is equal to the file's user
ID or (uid_t)−1 and group is equal either to the calling process' effective group ID or to one of
its supplementary group IDs.

EX If the path argument refers to a regular file, the set-user-ID (S_ISUID) and set-group-ID
(S_ISGID) bits of the file mode are cleared upon successful return from chown(), unless the call is
made by a process with appropriate privileges, in which case it is implementation-dependent
whether these bits are altered. If chown() is successfully invoked on a file that is not a regular
file, these bits may be cleared. These bits are defined in <sys/stat.h>.

EX If owner or group is specified as (uid_t)−1 or (gid_t)−1 respectively, the corresponding ID of the
file is unchanged.

Upon successful completion, chown() will mark for update the st_ctime field of the file.

RETURN VALUE
Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to indicate
the error. If −1 is returned, no changes are made in the user ID and group ID of the file.

ERRORS
The chown() function will fail if:
[EACCES] Search permission is denied on a component of the path prefix.
[EINVAL] Too many symbolic links were encountered in resolving path.
[ENAMETOOLONG] The length of the path argument exceeds [PATH_MAX] or a pathname
component is longer than [NAME_MAX].
[ENOTDIR] A component of the path prefix is not a directory.
[ENOENT] A component of path does not name an existing file or path is an empty string.
[EPERM] The effective user ID does not match the owner of the file, or the calling
process does not have appropriate privileges.
[EROFS] The named file resides on a read-only file system.

The chown() function may fail if:
[EIO] An I/O error occurred while reading or writing to the file system.
[EINTR] The chown() function was interrupted by a signal which was caught.
System Interface & Headers Issue 4, Version 2

chown()

EINVAL] The owner or group ID supplied is not a value supported by the implementation.

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds [PATH_MAX].

APPLICATION USAGE
Because {POSIX_CHOWN_RESTRICTED} is always defined with a value other than −1 on XSI-conformant systems, the error [EPERM] is always returned if the effective user ID does not match the owner of the file, or the calling process does not have appropriate privileges.

SEE ALSO
chown(), <sys/types.h>, <unistd.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:
• The type of argument path is changed from char * to const char *.

The following changes are incorporated for alignment with the FIPS requirements:
• In the ERRORS section, the condition whereby [ENAMETOOLONG] will be returned if a pathname component is larger that [NAME_MAX] is now defined as mandatory and marked as an extension.
• In the ERRORS section, the condition whereby [EPERM] will be returned when an attempt is made to change the user ID of a file and the caller does not have appropriate privileges is now defined as mandatory and marked as an extension.

Other changes are incorporated as follows:
• The header <sys/types.h> is now marked as optional (OH); this header need not be included on XSI-conformant systems.
• The value for owner of (uid_t)–1 is added to the DESCRIPTION section to allow the use of –1 by the owner of a file to change the group ID only.
• The APPLICATION USAGE section is added.

Issue 4, Version 2
The ERRORS section is updated for X/OPEN UNIX conformance as follows:
• It states that [ELOOP] will be returned if too many symbolic links are encountered during pathname resolution.
• The [EIO] and [EINTR] optional conditions are added.
• A second [ENAMETOOLONG] condition is defined that may report excessive length of an intermediate result of pathname resolution of a symbolic link.
NAME
chroot — change root directory (TO BE WITHDRAWN)

SYNOPSIS

```c
#include <unistd.h>

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

DESCRIPTION
The `path` argument points to a pathname naming a directory. The `chroot()` function causes the named directory to become the root directory, that is the starting point for path searches for pathnames beginning with `/`. The process’ working directory is unaffected by `chroot()`.

The process must have appropriate privileges to change the root directory.

The dot-dot entry in the root directory is interpreted to mean the root directory itself. Thus, dot-dot cannot be used to access files outside the subtree rooted at the root directory.

RETURN VALUE
Upon successful completion, 0 is returned. Otherwise, −1 is returned and `errno` is set to indicate the error. If −1 is returned, no change is made in the root directory.

ERRORS
The `chroot()` function will fail if:

- [EACCES] Search permission is denied for a component of `path`.
- [ELOOP] Too many symbolic links were encountered in resolving `path`.
- [ENAMETOOLONG] The length of the `path` argument exceeds `PATH_MAX` or a pathname component is longer than `NAME_MAX`.
- [ENOENT] A component of `path` does not name an existing directory or `path` is an empty string.
- [ENOTDIR] A component of the `path` name is not a directory.
- [EPERM] The effective user ID does not have appropriate privileges.

The `chroot()` function may fail if:

- [ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds `PATH_MAX`.

APPLICATION USAGE
There is no portable use that an application could make of this interface. It is therefore marked TO BE WITHDRAWN.

SEE ALSO
`chdir()`, `<unistd.h>`.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.
Issue 4

Changes are incorporated as follows:

• The interface is marked TO BE WITHDRAWN, as there is no portable use that an application could make of this interface.

• The header `<unistd.h>` is added to the **SYNOPSIS** section.

• The type of argument `path` is changed from `char *` to `const char *`.

• The **APPLICATION USAGE** section is added.

• The **DESCRIPTION** section now refers to the process' working directory instead of the user's working directory.

Issue 4, Version 2

The **ERRORS** section is updated for X/OPEN UNIX conformance as follows:

• It states that `[ELOOP]` will be returned if too many symbolic links are encountered during pathname resolution.

• A second `[ENAMETOOLONG]` condition is defined that may report excessive length of an intermediate result of pathname resolution of a symbolic link.
NAME
clearerr — clear indicators on a stream

SYNOPSIS
#include <stdio.h>
void clearerr(FILE *stream);

DESCRIPTION
The clearerr() function clears the end-of-file and error indicators for the stream to which stream points.

RETURN VALUE
The clearerr() function returns no value.

ERRORS
No errors are defined.

SEE ALSO
<stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.
NAME
clock — report CPU time used

SYNOPSIS
#include <time.h>
clock_t clock(void);

DESCRIPTION
The clock() function returns the implementation's best approximation to the processor time used by the process since the beginning of an implementation-dependent time related only to the process invocation.

RETURN VALUE
To determine the time in seconds, the value returned by clock() should be divided by the value of the macro CLOCKS_PER_SEC. CLOCKS_PER_SEC is defined to be one million in <time.h>. If the processor time used is not available or its value cannot be represented, the function returns the value (clock_t)−1.

ERRORS
No errors are defined.

SEE ALSO
asctime(), ctime(), difftime(), gmtime(), localtime(), mktime(), strftime(), strptime(), time(), utime(), <time.h>.

APPLICATION USAGE
In order to measure the time spent in a program, clock() should be called at the start of the program and its return value subtracted from the value returned by subsequent calls. The value returned by clock() is defined for compatibility across systems that have clocks with different resolutions. The resolution on any particular system may not be to microsecond accuracy.

The value returned by clock() may wrap around on some systems. For example, on a machine with 32-bit values for clock_t, it will wrap after 2147 seconds or 36 minutes.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated for alignment with the ISO C standard:

• The header <time.h> is added to the SYNOPSIS section.

• The DESCRIPTION and RETURN VALUE sections, though functionally equivalent to Issue 3, are rewritten for clarity and consistency with the ISO C standard. This issue also defines under what circumstances (clock_t)−1 can be returned by the function.

• The function is no longer marked as an extension.

Other changes are incorporated as follows:

• Reference to the resolution of CLOCKS_PER_SEC is marked as an extension.

• The ERRORS section is added.

• Advice on how to calculate the time spent in a program is added to the APPLICATION USAGE section.
NAME

close — close a file descriptor

SYNOPSIS

#include <unistd.h>

int close(int fildes);

DESCRIPTION

The close() function will deallocate the file descriptor indicated by fildes. To deallocate means to
make the file descriptor available for return by subsequent calls to open() or other functions that
allocate file descriptors. All outstanding record locks owned by the process on the file
associated with the file descriptor will be removed (that is, unlocked).

If close() is interrupted by a signal that is to be caught, it will return −1 with errno set to [EINTR]
and the state of fildes is unspecified.

When all file descriptors associated with a pipe or FIFO special file are closed, any data
remaining in the pipe or FIFO will be discarded.

When all file descriptors associated with an open file description have been closed the open file
description will be freed.

If the link count of the file is 0, when all file descriptors associated with the file are closed, the
space occupied by the file will be freed and the file will no longer be accessible.

UX

If a STREAMS-based fildes is closed and the calling process was previously registered to receive
a SIGPOLL signal for events associated with that STREAM, the calling process will be
unregistered for events associated with the STREAM. The last close() for a STREAM causes the
STREAM associated with fildes to be dismantled. If O_NONBLOCK is not set and there have
been no signals posted for the STREAM, and if there is data on the module's write queue, close() waits for an unspecified time (for each module and driver) for any output to drain before
dismantling the STREAM. The time delay can be changed via an I_SETCLTIME ioctl() request.
If the O_NONBLOCK flag is set, or if there are any pending signals, close() does not wait for
output to drain, and dismantles the STREAM immediately.

If the implementation supports STREAMS-based pipes, and fildes is associated with one end of a
pipe, the last close() causes a hangup to occur on the other end of the pipe. In addition, if the
other end of the pipe has been named by fattach(), then the last close() forces the named end to
be detached by fdetach(). If the named end has no open file descriptors associated with it and
gets detached, the STREAM associated with that end is also dismantled.

If fildes refers to the master side of a pseudo-terminal, a SIGHUP signal is sent to the process
group, if any, for which the slave side of the pseudo-terminal is the controlling terminal. It is
unspecified whether closing the master side of the pseudo-terminal flushes all queued input and
output.

If fildes refers to the slave side of a STREAMS-based pseudo-terminal, a zero-length message
may be sent to the master.

RETURN VALUE

Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to indicate
the error.

ERRORS

The close() function will fail if:

[EBADDF] The fildes argument is not a valid file descriptor.
[EINTR] The close() function was interrupted by a signal.

UX The close() function may fail if:

[EIO] An I/O error occurred while reading from or writing to the file system.

APPLICATION USAGE
An application that had used the stdio routine fopen() to open a file should use the corresponding fclose() routine rather than close().

SEE ALSO
fattach(), fclose(), fdetach(), fopen(), ioctl(), open(), <unistd.h>, Section 2.5 on page 35.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:

• The header <unistd.h> is added to the SYNOPSIS section.

Issue 4, Version 2
The following changes are incorporated for X/OPEN UNIX conformance:

• The DESCRIPTION is updated to describe the actions of closing a file descriptor referring to a STREAMS-based file or either side of a pseudo-terminal.

• The ERRORS section describes a condition under which the [EIO] error may be returned.
closedir()

NAME
closedir — close a directory stream

SYNOPSIS
#include <sys/types.h>
#include <dirent.h>
int closedir(DIR * dirp);

DESCRIPTION
The closedir() function closes the directory stream referred to by the argument dirp. Upon return, the value of dirp may no longer point to an accessible object of the type DIR. If a file descriptor is used to implement type DIR, that file descriptor will be closed.

RETURN VALUE
Upon successful completion, closedir() returns 0. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS
The closedir() function may fail if:

[EBADF] The dirp argument does not refer to an open directory stream.

EX [EINTR] The closedir() function was interrupted by a signal.

SEE ALSO
opendir(), <dirent.h>, <sys/types.h>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The following changes are incorporated in this issue:

• The header <sys/types.h> is now marked as optional (OH); this header need not be included on XSI-conformant systems.

• The [EINTR] error is marked as an extension.
NAME
closelog, openlog, setlogmask, syslog — control system log

SYNOPSIS
UX
#include <syslog.h>

void closelog(void);

void openlog(const char *ident, int logopt, int facility);

int setlogmask(int maskpri);

void syslog(int priority, const char *message, ... /* arguments */);

DESCRIPTION
The syslog() function sends a message to a logging facility, which logs it in an appropriate
system log, writes it to the system console, forwards it to a list of users, or forwards it to the
logging facility on another host over the network. The logged message includes a message
header and a message body. The message header consists of a facility indicator, a severity level
indicator, a timestamp, a tag string, and optionally the process ID.

The message body is generated from the message and following arguments in the same manner
as if these were arguments to printf(), except that occurrences of %m in the format string
pointed to by the message argument are replaced by the error message string associated with the
current value of errno. A trailing newline character is added if needed.

Values of the priority argument are formed by ORing together a severity level value and an
optional facility value. If no facility value is specified, the current default facility value is used.

Possible values of severity level include:

LOG_EMERG      A panic condition. This is normally broadcast to each login.
LOG_ALERT      A condition that should be corrected immediately, such as a corrupted system
database.
LOG_CRIT       Critical conditions, such as hard device errors.
LOG_ERR        Errors.
LOG_WARNING    Warning messages.
LOG_NOTICE     Conditions that are not error conditions, but that may require special
handling.
LOG_INFO       Informational messages.
LOG_DEBUG      Messages that contain information normally of use only when debugging a
program.

The facility indicates the application or system component generating the message. Possible
facility values include:

LOG_USER       Messages generated by random processes. This is the default facility identifier
if none is specified.
LOG_LOCAL0     Reserved for local use.
LOG_LOCAL1     Reserved for local use.
LOG_LOCAL2     Reserved for local use.
LOG_LOCAL3 Reserved for local use.
LOG_LOCAL4 Reserved for local use.
LOG_LOCAL5 Reserved for local use.
LOG_LOCAL6 Reserved for local use.
LOG_LOCAL7 Reserved for local use.

The `openlog()` function sets process attributes that affect subsequent calls to `syslog()`. The `ident` argument is a string that is prepended to every message. The `logopt` argument indicates logging options. Values for `logopt` are constructed by a bitwise-inclusive OR of zero or more of the following:

- **LOG_PID** Log the process ID with each message. This is useful for identifying specific processes.
- **LOG_CONS** Write messages to the system console if they cannot be sent to the logging facility. This option is safe to use in processes that have no controlling terminal, since `syslog()` forks before opening the console.
- **LOG_NDELAY** Open the connection to the logging facility immediately. Normally the open is delayed until the first message is logged. This is useful for programs that need to manage the order in which file descriptors are allocated.
- **LOG_ODELAY** Delay open until `syslog()` is called.
- **LOG_NOWAIT** Do not wait for child processes that have been forked to log messages onto the console. This option should be used by processes that enable notification of child termination using SIGCHLD, since `syslog()` may otherwise block waiting for a child whose exit status has already been collected.

The `facility` argument encodes a default facility to be assigned to all messages that do not have an explicit facility already encoded. The initial default facility is LOG_USER.

The `openlog()` and `syslog()` functions may allocate a file descriptor. It is not necessary to call `openlog()` prior to calling `syslog()`.

The `closelog()` function closes any open file descriptors allocated by previous calls to `openlog()` or `syslog()`.

The `setlogmask()` function sets the log priority mask for the current process to `maskpri` and returns the previous mask. If the `maskpri` argument is 0, the current log mask is not modified. Calls by the current process to `syslog()` with a priority not set in `maskpri` are rejected. The mask for an individual priority `pri` is calculated by the macro LOG_MASK(pri); the mask for all priorities up to and including `toppri` is given by the macro LOG_UPTO(toppri). The default log mask allows all priorities to be logged.

Symbolic constants for use as values of the `logopt`, `facility`, `priority`, and `maskpri` arguments are defined in the `<syslog.h>` header.

**RETURN VALUE**

The `setlogmask()` function returns the previous log priority mask. The `closelog()`, `openlog()` and `syslog()` functions return no value.

**ERRORS**

No errors are defined.

**SEE ALSO**

`printf()`, `<syslog.h>`.
CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
compile — produce compiled regular expression (TO BE WITHDRAWN)

SYNOPSIS
EX
#include <regexp.h>
char *compile(char *instring, char *expbuf, const char *endbuf, int eof);

DESCRIPTION
Refer to regexp().

CHANGE HISTORY
First released in Issue 2.
Derived from Issue 2 of the SVID.

Issue 4
The following changes are incorporated in this issue:

• The header <regexp.h> is added to the SYNOPSIS section.
• The type of argument endbuf is changed from char * to const char *.
• The interface is marked TO BE WITHDRAWN, because improved functionality is now provided by interfaces introduced for alignment with the ISO POSIX-2 standard.
NAME
  confstr — get configurable variables

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

DESCRIPTION
  The confstr() function provides a method for applications to get configuration-defined string
  values. Its use and purpose are similar to sysconf(), but it is used where string values rather than
  numeric values are returned.

  The name argument represents the system variable to be queried. The implementation supports
  the name value of _CS_PATH, defined in <unistd.h>. It may support others.

  If len is not 0, and if name has a configuration-defined value, confstr() copies that value into the
  len-byte buffer pointed to by buf. If the string to be returned is longer than len bytes, including
  the terminating null, then confstr() truncates the string to len−1 bytes and null-terminates the
  result. The application can detect that the string was truncated by comparing the value returned
  by confstr() with len.

  If len is 0 and buf is a null pointer, then confstr() still returns the integer value as defined below,
  but does not return a string. If len is 0 but buf is not a null pointer, the result is unspecified.

RETURN VALUE
  If name has a configuration-defined value, confstr() returns the size of buffer that would be
  needed to hold the entire configuration-defined value. If this return value is greater than len, the
  string returned in buf is truncated.

  If name is invalid, confstr() returns 0 and sets errno to indicate the error.

  If name does not have a configuration-defined value, confstr() returns 0 and leaves errno
  unchanged.

ERRORS
  The confstr() function will fail if:

  [EINVAL] The value of the name argument is invalid.

APPLICATION USAGE
  An application can distinguish between an invalid name parameter value and one that
  corresponds to a configurable variable that has no configuration-defined value by checking if
  errno is modified. This mirrors the behaviour of sysconf().

  The original need for this function was to provide a way of finding the configuration-defined
  default value for the environment variable PATH. Since PATH can be modified by the user to
  include directories that could contain utilities replacing XCU specification standard utilities,
  applications need a way to determine the system-supplied PATH environment variable value
  that contains the correct search path for the standard utilities.

  An application could use:

  confstr(name, (char *)NULL, (size_t)0)

  to find out how big a buffer is needed for the string value; use malloc() to allocate a buffer to
  hold the string; and call confstr() again to get the string. Alternately, it could allocate a fixed,
  static buffer that is big enough to hold most answers (perhaps 512 or 1024 bytes), but then use
  malloc() to allocate a larger buffer if it finds that this is too small.
SEE ALSO

pathconf(), sysconf(), <unistd.h>, <regexp.h>, the XCU specification.

CHANGE HISTORY

First released in Issue 4.

Derived from the ISO POSIX-2 standard.
NAME

cos — cosine function

SYNOPSIS

#include <math.h>

double cos(double x);

DESCRIPTION

The `cos()` function computes the cosine of \( x \), measured in radians.

RETURN VALUE

Upon successful completion, `cos()` returns the cosine of \( x \).

EX

If \( x \) is NaN, NaN is returned and \( errno \) may be set to [EDOM].

EX

If \( x \) is ±Inf, either 0 is returned and \( errno \) is set to [EDOM], or NaN is returned and \( errno \) may be set to [EDOM].

If the result underflows, 0 is returned and \( errno \) may be set to [ERANGE].

ERRORS

The `cos()` function may fail if:

EX

[EDOM] The value of \( x \) is NaN or \( x \) is ±Inf.

[ERANGE] The result underflows.

EX

No other errors will occur.

APPLICATION USAGE

An application wishing to check for error situations should set \( errno \) to 0 before calling `cos()`. If \( errno \) is non-zero on return, or the returned value is NaN, an error has occurred.

The `cos()` function may lose accuracy when its argument is far from 0.

SEE ALSO

`acos()`, `isnan()`, `sin()`, `tan()`, `<math.h>`.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated in this issue:

- Removed references to `matherr()`.
- The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the ISO C standard and to rationalise error handling in the mathematics functions.
- The return value specified for [EDOM] is marked as an extension.
cosh() NAME  
cosh — hyperbolic cosine function

SYNOPSIS  
#include <math.h>

double cosh(double x);

DESCRIPTION  
The cosh() function computes the hyperbolic cosine of x.

RETURN VALUE  
Upon successful completion, cosh() returns the hyperbolic cosine of x.

If the result would cause an overflow, HUGE_VAL is returned and errno is set to [ERANGE].

EX  
If x is NaN, NaN is returned and errno may be set to [EDOM].

ERRORS  
The cosh() function will fail if:

[ERANGE] The result would cause an overflow.

The cosh() function may fail if:

EX [EDOM] The value of x is NaN.

EX No other errors will occur.

APPLICATION USAGE  
An application wishing to check for error situations should set errno to 0 before calling cosh(). If errno is non-zero on return, or the returned value is NaN, an error has occurred.

SEE ALSO  
acosh(), isnan(), sinh(), tanh(), <math.h>.

CHANGE HISTORY  
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4  
The following changes are incorporated in this issue:

• Removed references to matherr().

• The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the ISO C standard and to rationalise error handling in the mathematics functions.

• The return value specified for [EDOM] is marked as an extension.
NAME
creat — create a new file or rewrite an existing one

SYNOPSIS
OH
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

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

DESCRIPTION
The function call:
    creat(path, mode)

is equivalent to:
    open(path, O_WRONLY|O_CREAT|O_TRUNC, mode)

RETURN VALUE
Refer to open().

ERRORS
Refer to open().

SEE ALSO
open(), <fcntl.h>, <sys/stat.h>, <sys/types.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:
- The type of argument path is changed from char * to const char *.

Other changes are incorporated as follows:
- The headers <sys/types.h> and <sys/stat.h> are now marked as optional (OH); these headers need not be included on XSI-conformant systems.
NAME

crypt — string encoding function

SYNOPSIS

```
#include <unistd.h>

char *crypt (const char *key, const char *salt);
```

DESCRIPTION

The `crypt()` function is a string encoding function. The algorithm is implementation-dependent.

The `key` argument points to a string to be encoded. The `salt` argument is a string chosen from the set:

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789./
```

The first two characters of this string may be used to perturb the encoding algorithm.

RETURN VALUE

Upon successful completion, `crypt()` returns a pointer to the encoded string. The first two characters of the returned value are those of the `salt` argument.

Otherwise it returns a null pointer and sets `errno` to indicate the error.

ERRORS

The `crypt()` function will fail if:

- [ENOSYS] The functionality is not supported on this implementation.

APPLICATION USAGE

The return value of `crypt()` points to static data that is overwritten by each call.

The values returned by this function may not be portable among XSI-conformant systems.

SEE ALSO

`encrypt()`, `setkey()`, `<unistd.h>`.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated in this issue:

- The header `<unistd.h>` is added to the SYNOPSIS section.
- The type of arguments `key` and `salt` are changed from `char *` to `const char *`.
- The DESCRIPTION section now explicitly defines the characters that can appear in the `salt` argument.
NAME
ctermid — generate pathname for controlling terminal

SYNOPSIS
#include <stdio.h>
char *ctermid(char * s);

DESCRIPTION
The ctermid() function generates a string that, when used as a pathname, refers to the current controlling terminal for the current process. If ctermid() returns a pathname, access to the file is not guaranteed.

RETURN VALUE
If s is a null pointer, the string is generated in an area that may be static (and therefore may be overwritten by each call), the address of which is returned. Otherwise s is assumed to point to a character array of at least {L_ctermid} bytes; the string is placed in this array and the value of s is returned. The symbolic constant {L_ctermid} is defined in <stdio.h>, and will have a value greater than 0.

The ctermid() function will return an empty string if the pathname that would refer to the controlling terminal cannot be determined, or if the function is unsuccessful.

ERRORS
No errors are defined.

APPLICATION USAGE
The difference between ctermid() and ttyname() is that ttyname() must be handed a file descriptor and returns a path of the terminal associated with that file descriptor, while ctermid() returns a string (such as /dev/tty) that will refer to the current controlling terminal if used as a pathname.

SEE ALSO
ttyname(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:
- The DESCRIPTION and RETURN VALUE sections, though functionally identical to Issue 3, are rewritten.
NAME
ctime — convert time value to date and time string

SYNOPSIS
#include <time.h>
char *ctime(const time_t *clock);

DESCRIPTION
The ctime() function converts the time pointed to by clock, representing time in seconds since the Epoch, to local time in the form of a string. It is equivalent to:
asci time(localtime(clock))

RETURN VALUE
The ctime() function returns the pointer returned by asctime() with that broken-down time as an argument.

ERRORS
No errors are defined.

APPLICATION USAGE
The asctime(), ctime(), gmtime() and localtime() functions return values in one of two static objects: a broken-down time structure and an array of char. Execution of any of the functions may overwrite the information returned in either of these objects by any of the other functions.

Values for the broken-down time structure can be obtained by calling gmtime() or localtime(). This interface is included for compatibility with older implementations, and does not support localised date and time formats. Applications should use the strftime() interface to achieve maximum portability.

SEE ALSO
asctime(), clock(), difftime(), gmtime(), localtime(), mktime(), strftime(), strptime(), time(), utime(), <time.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
- The type of argument clock is changed from time_t* to const time_t*.

Another change is incorporated as follows:
- The APPLICATION USAGE section is expanded to describe the time-handling functions generally and to refer users to strftime(), which is a locale-dependent time-handling function.
NAME
cuserid — character login name of the user (TO BE WITHDRAWN)

SYNOPSIS
EX
#include <stdio.h>
char *cuserid(char *s);

DESCRIPTION
The cuserid() function generates a character representation of the name associated with the real or effective user ID of the process.

If s is a null pointer, this representation is generated in an area that may be static (and thus overwritten by subsequent calls to cuserid()), the address of which is returned. If s is not a null pointer, s is assumed to point to an array of at least \[L_cuserid\] bytes; the representation is deposited in this array. The symbolic constant \[L_cuserid\] is defined in <stdio.h> and has a value greater than 0.

RETURN VALUE
If s is not a null pointer, s is returned. If s is not a null pointer and the login name cannot be found, the null byte ‘\0’ will be placed at ‘s’. If s is a null pointer and the login name cannot be found, cuserid() returns a null pointer. If s is a null pointer and the login name can be found, the address of a buffer (possibly static) containing the login name is returned.

ERRORS
No errors are defined.

APPLICATION USAGE
The functionality of cuserid() defined in the POSIX.1-1988 standard (and Issue 3 of this document) differs from that of historical implementations (and Issue 2 of this document). In the ISO POSIX-1 standard, cuserid() is removed completely. In this document, therefore, both functionalities are allowed, but both are also marked TO BE WITHDRAWN.

The Issue 2 functionality can be obtained by using:
getpwuid(getuid())

The Issue 3 functionality can be obtained by using:
getpwuid(geteuid())

SEE ALSO
getlogin(), getpwnam(), getpwuid(), getuid(), geteuid(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from System V Release 2.0.

Issue 4
The following changes are incorporated in this issue:
• The interface is marked TO BE WITHDRAWN, because of differences between the historical definition of this interface and the definition published in the POSIX.1-1988 standard (and hence Issue 3). The interface has also been removed from the ISO POSIX-1 standard.
• The interface is now marked as an extension.
• The DESCRIPTION section is changed to indicate that an implementation can determine the name returned by the function from the real or effective user ID of the process.
• The **APPLICATION USAGE** section is rewritten to describe the historical development of this interface, and to indicate transition between this and previous issues.

• The **RETURN VALUE** section has been expanded.
NAME
    daylight — daylight savings time flag

SYNOPSIS
    #include <time.h>
    extern int daylight;

DESCRIPTION
    Refer to tzset().

CHANGE HISTORY
    First released in Issue 1.
    Derived from Issue 1 of the SVID.
NAME
dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey, dbm_nextkey,
dbm_open, dbm_store — database functions

SYNOPSIS

```
#include <ndbm.h>

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

DESCRIPTION

These functions create, access and modify a database.

A datum consists of at least two members, dptr and dsize. The dptr member points to an object that is dsize bytes in length. Arbitrary binary data, as well as character strings, may be stored in the object pointed to by dptr.

The database is stored in two files. One file is a directory containing a bit map of keys and has .dir as its suffix. The second file contains all data and has .pag as its suffix.

The dbm_open() function opens a database. The file argument to the function is the pathname of the database. The function opens two files named file.dir and file.pag. The open_flags argument has the same meaning as the flags argument of open() except that a database opened for write-only access opens the files for read and write access. The file_mode argument has the same meaning as the third argument of open().

The dbm_close() function closes a database. The argument db must be a pointer to a DBM structure that has been returned from a call to dbm_open().

The dbm_fetch() function reads a record from a database. The argument db is a pointer to a database structure that has been returned from a call to dbm_open(). The argument key is a datum that has been initialised by the application program to the value of the key that matches the key of the record the program is fetching.

The dbm_store() function writes a record to a database. The argument db is a pointer to a database structure that has been returned from a call to dbm_open(). The argument key is a datum that has been initialised by the application program to the value of the key that identifies (for subsequent reading, writing or deleting) the record the program is writing. The argument content is a datum that has been initialised by the application program to the value of the record the program is writing. The argument store_mode controls whether dbm_store() replaces any pre-existing record that has the same key that is specified by the key argument. The application program must set store_mode to either DBM_INSERT or DBM_REPLACE. If the database contains a record that matches the key argument and store_mode is DBM_REPLACE, the existing record is replaced with the new record. If the database contains a record that matches the key argument and store_mode is DBM_INSERT, the existing record is not replaced with the new record. If the database does not contain a record that matches the key argument and store_mode
is either DBM_INSERT or DBM_REPLACE, the new record is inserted in the database.

The `dbm_delete()` function deletes a record and its key from the database. The argument `db` is a pointer to a database structure that has been returned from a call to `dbm_open()`. The argument `key` is a `datum` that has been initialised by the application program to the value of the key that identifies the record the program is deleting.

The `dbm_firstkey()` function returns the first key in the database. The argument `db` is a pointer to a database structure that has been returned from a call to `dbm_open()`.

The `dbm_nextkey()` function returns the next key in the database. The argument `db` is a pointer to a database structure that has been returned from a call to `dbm_open()`. The `dbm_firstkey()` function must be called before calling `dbm_nextkey()`. Subsequent calls to `dbm_nextkey()` return the next key until all of the keys in the database have been returned.

The `dbm_error()` function returns the error condition of the database. The argument `db` is a pointer to a database structure that has been returned from a call to `dbm_open()`.

The `dbm_clearerr()` function clears the error condition of the database. The argument `db` is a pointer to a database structure that has been returned from a call to `dbm_open()`.

These database functions support key/content pairs of at least 1024 bytes.

RETURN VALUE
The `dbm_store()` and `dbm_delete()` functions return 0 when they succeed and a negative value when they fail.

The `dbm_store()` function returns 1 if it is called with a `flags` value of DBM_INSERT and the function finds an existing record with the same key.

The `dbm_error()` function returns 0 if the error condition is not set and returns a non-zero value if the error condition is set.

The return value of `dbm_clearerr()` is unspecified.

The `dbm_firstkey()` and `dbm_nextkey()` functions return a key `datum`. When the end of the database is reached, the `dptr` member of the key is a null pointer. If an error is detected, the `dptr` member of the key is a null pointer and the error condition of the database is set.

The `dbm_fetch()` function returns a content `datum`. If no record in the database matches the key or if an error condition has been detected in the database, the `dptr` member of the content is a null pointer.

The `dbm_open()` function returns a pointer to a database structure. If an error is detected during the operation, `dbm_open()` returns a `(DBM *)0`.

ERRORS
No errors are defined.

APPLICATION USAGE
The following code can be used to traverse the database:

```c
for(key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))
```

The `dbm_` functions provided in this library should not be confused in any way with those of a general-purpose database management system. These functions do not provide for multiple search keys per entry, they do not protect against multi-user access (in other words they do not lock records or files), and they do not provide the many other useful database functions that are found in more robust database management systems. Creating and updating databases by use of these functions is relatively slow because of data copies that occur upon hash collisions. These functions are useful for applications requiring fast lookup of relatively static information.
that is to be indexed by a single key.

The `dptr` pointers returned by these functions may point into static storage that may be changed by subsequent calls.

The `dbm_delete()` function does not physically reclaim file space, although it does make it available for reuse.

After calling `dbm_store()` or `dbm_delete()` during a pass through the keys by `dbm_firstkey()` and `dbm_nextkey()`, the application should reset the database by calling `dbm_firstkey()` before again calling `dbm_nextkey()`.

SEE ALSO
`open()`, `<ndbm.h>`.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
difftime — compute the difference between two calendar time values

SYNOPSIS
#include <time.h>

double difftime(time_t time1, time_t time0);

DESCRIPTION
The difftime() function computes the difference between two calendar times (as
returned by time()): time1 – time0.

RETURN VALUE
The difftime() function returns the difference expressed in seconds as a type double.

ERRORS
No errors are defined.

SEE ALSO
asctime(), clock(), ctime(), gmtime(), localtime(), mktime(), strftime(), strptime(),
time(), utime(), <time.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the ISO C standard.
NAME
dirname — report the parent directory name of a file pathname

SYNOPSIS
UX

```c
#include <libgen.h>

char *dirname(char *path);
```

DESCRIPTION
The `dirname()` function takes a pointer to a character string that contains a pathname, and returns a pointer to a string that is a pathname of the parent directory of that file. Trailing '/' characters in the path are not counted as part of the path.

If `path` does not contain a '/', then `dirname()` returns a pointer to the string "." . If `path` is a null pointer or points to an empty string, `dirname()` returns a pointer to the string "." .

RETURN VALUE
The `dirname()` function returns a pointer to a string that is the parent directory of `path`. If `path` is a null pointer or points to an empty string, a pointer to a string "." is returned.

ERRORS
No errors are defined.

EXAMPLES

<table>
<thead>
<tr>
<th>Input String</th>
<th>Output String</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;/usr/lib&quot;</td>
<td>&quot;/usr&quot;</td>
</tr>
<tr>
<td>&quot;/usr/&quot;</td>
<td>&quot;/&quot;</td>
</tr>
<tr>
<td>&quot;usr&quot;</td>
<td>&quot;.&quot;</td>
</tr>
<tr>
<td>&quot;/&quot;</td>
<td>&quot;/&quot;</td>
</tr>
<tr>
<td>&quot;.&quot;</td>
<td>&quot;.&quot;</td>
</tr>
<tr>
<td>&quot;..&quot;</td>
<td>&quot;.&quot;</td>
</tr>
</tbody>
</table>

The following code fragment reads a pathname, changes the current working directory to the parent directory, and opens the file.

```c
char path[MAXPATHLEN], *pathcopy;
int fd;
fgets(path, MAXPATHLEN, stdin);
pathcopy = strdup(path);  
chdir(dirname(pathcopy));
fd = open(basename(path), O_RDONLY);
```

APPLICATION USAGE
The `dirname()` function may modify the string pointed to by `path`, and may return a pointer to static storage that may then be overwritten by subsequent calls to `dirname()`.

The `dirname()` and `basename()` functions together yield a complete pathname. The expression `dirname(path)` obtains the pathname of the directory where `basename(path)` is found.

SEE ALSO
`basename()`, `<libgen.h>`.

CHANGE HISTORY
First released in Issue 4, Version 2.
div() (System Interfaces)

NAME

div — compute quotient and remainder of an integer division

SYNOPSIS

#include <stdlib.h>

div_t div(int numer, int denom);

DESCRIPTION

The div() function computes the quotient and remainder of the division of the numerator numer by the denominator denom. If the division is inexact, the resulting quotient is the integer of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be represented, the behaviour is undefined; otherwise, quot * denom + rem will equal numer.

RETURN VALUE

The div() function returns a structure of type div_t, comprising both the quotient and the remainder. The structure includes the following members, in any order:

    int quot; /* quotient */
    int rem; /* remainder */

ERRORS

No errors are defined.

SEE ALSO

ldiv(), <stdlib.h>.

CHANGE HISTORY

First released in Issue 4.

Derived from the ISO C standard.
NAME
drand48, erand48, jrand48, lcong48, lrand48, mrand48, nrand48, seed48, srand48 — generate uniformly distributed pseudo-random numbers

SYNOPSIS
#include <stdlib.h>

double drand48(void);
double erand48(unsigned short int xsubi[3]);
long int jrand48(unsigned short int xsubi[3]);
void lcong48(unsigned short int param[7]);
long int lrand48(void);
long int mrand48(void);
long int nrand48(unsigned short int xsubi[3]);
unsigned short int *seed48(unsigned short int seed16v[3]);
void srand48(long int seedval);

DESCRIPTION
This family of functions generates pseudo-random numbers using a linear congruential algorithm and 48-bit integer arithmetic.

The drand48() and erand48() functions return non-negative, double-precision, floating-point values, uniformly distributed over the interval \([0.0,1.0)\).

The lrand48() and nrand48() functions return non-negative, long integers, uniformly distributed over the interval \([0,2^{31})\).

The mrand48() and jrand48() functions return signed long integers uniformly distributed over the interval \([-2^{31},2^{31})\).

The srand48(), seed48() and lcong48() are initialisation entry points, one of which should be invoked before either drand48(), lrand48() or mrand48() is called. (Although it is not recommended practice, constant default initialiser values will be supplied automatically if drand48(), lrand48() or mrand48() is called without a prior call to an initialisation entry point.) The erand48(), nrand48() and jrand48() functions do not require an initialisation entry point to be called first.

All the routines work by generating a sequence of 48-bit integer values, \(X_n\), according to the linear congruential formula:

\[ X_{n+1} = (aX_n + c) \mod m \quad n \geq 0 \]

The parameter \(m = 2^{48}\), hence 48-bit integer arithmetic is performed. Unless lcong48() is invoked, the multiplier value \(a\) and the addend value \(c\) are given by:

\[ a = 5DEECE66D_{16} = 273673163155_8 \]
\[ c = B_{16} = 13_8 \]

The value returned by any of the drand48(), erand48(), jrand48(), lrand48(), mrand48() or nrand48() functions is computed by first generating the next 48-bit \(X_n\) in the sequence. Then the appropriate number of bits, according to the type of data item to be returned, are copied from the high-order (leftmost) bits of \(X_n\) and transformed into the returned value.
The `drand48()`, `lrand48()` and `mrand48()` functions store the last 48-bit \( X_i \) generated in an internal buffer; that is why they must be initialised prior to being invoked. The `erand48()`, `nrand48()` and `jrand48()` functions require the calling program to provide storage for the successive \( X_i \) values in the array specified as an argument when the functions are invoked. By using different arguments, `erand48()`, `nrand48()` and `jrand48()` allow separate modules of a large program to generate several independent streams of pseudo-random numbers, that is the sequence of numbers in each stream will not depend upon how many times the routines are called to generate numbers for the other streams.

The initialiser function `srand48()` sets the high-order 32 bits of \( X_i \) to the low-order 32 bits contained in its argument. The low-order 16 bits of \( X_i \) are set to the arbitrary value 330E16.

The initialiser function `seed48()` sets the value of \( X_i \) to the 48-bit value specified in the argument array. The low-order 16 bits of \( X_i \) are set to the low-order 16 bits of `seed16v[0]`. The mid-order 16 bits of \( X_i \) are set to the low-order 16 bits of `seed16v[1]`. The high-order 16 bits of \( X_i \) are set to the low-order 16 bits of `seed16v[2]`. In addition, the previous value of \( X_i \) is copied into a 48-bit internal buffer, used only by `seed48()`, and a pointer to this buffer is the value returned by `seed48()`. This returned pointer, which can just be ignored if not needed, is useful if a program is to be restarted from a given point at some future time — use the pointer to get at and store the last \( X_i \) value, and then use this value to reinitialise via `seed48()` when the program is restarted.

The initialiser function `lcong48()` allows the user to specify the initial \( X_i \), the multiplier value \( a \), and the addend value \( c \). Argument array elements `param[0-2]` specify \( X_i \), `param[3-5]` specify the multiplier \( a \), and `param[6]` specifies the 16-bit addend \( c \). After `lcong48()` is called, a subsequent call to either `srand48()` or `seed48()` will restore the “standard” multiplier and addend values, \( a \) and \( c \), specified above.

RETURN VALUE
As described in the DESCRIPTION section above.

ERRORS
No errors are defined.

SEE ALSO
`rand()`, `<stdlib.h>`.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

- The type `long` is replaced by `long int` and the type `unsigned short` is replaced by `unsigned short int` in the SYNOPSIS section.

- In the DESCRIPTION section, the description of `srand48()` is amended to fix a limitation in Issue 3, which indicates that the high-order 32 bits of \( X_i \) are set to the {LONG_BIT} bits in the argument. Though unintentional, the implication of this statement is that {LONG_BIT} would be 32 on all systems compliant with Issue 3, when in fact Issue 3 imposes no such restriction.

- The header `<stdlib.h>` is added to the SYNOPSIS section.
NAME
dup, dup2 — duplicate an open file descriptor

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

DESCRIPTION
The dup() and dup2() functions provide an alternative interface to the service provided by
fcntl() using the F_DUPFD command. The call:

    fid = dup(fildes);

is equivalent to:

    fid = fcntl(fildes, F_DUPFD, 0);

The call:

    fid = dup2(fildes, fildes2);

is equivalent to:

    close(fildes2);
    fid = fcntl(fildes, F_DUPFD, fildes2);

except for the following:

- If fildes2 is less than 0 or greater than or equal to {OPEN_MAX}, dup2() returns −1 with errno
  set to [EBADF].
- If fildes is a valid file descriptor and is equal to fildes2, dup2() returns fildes2 without closing it.
- If fildes is not a valid file descriptor, dup2() returns −1 and does not close fildes2.
- The value returned is equal to the value of fildes2 upon successful completion, or is −1 upon
  failure.

RETURN VALUE
Upon successful completion a non-negative integer, namely the file descriptor, is returned. Otherwise,
−1 is returned and errno is set to indicate the error.

ERRORS
The dup() function will fail if:

- [EBADF] The fildes argument is not a valid open file descriptor.
- [EMFILE] The number of file descriptors in use by this process would exceed
  {OPEN_MAX}.

The dup2() function will fail if:

- [EBADF] The fildes argument is not a valid open file descriptor or the argument fildes2 is
  negative or greater than or equal to {OPEN_MAX}.
- [EINTR] The dup2() function was interrupted by a signal.

SEE ALSO
close(), fcntl(), open(), <unistd.h>.
CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated for alignment with the ISO POSIX-1 standard:

• In the DESCRIPTION, the fourth bullet item describing differences between \texttt{dup()} and \texttt{dup2()} is added.

• In the ERRORS section, error values returned by \texttt{dup()} and \texttt{dup2()} are now described separately.

Other changes are incorporated as follows:

• The header \texttt{<unistd.h>} is added to the SYNOPSIS section.

• [EINTR] is no longer required for \texttt{dup()} because \texttt{fcntl()} does not return [EINTR] for F_DUPFD.
NAME
ecvt, fcvt, gcvt — convert floating-point number to string

SYNOPSIS

UX
#include <stdlib.h>

char *ecvt(double value, int ndigit, int *decpt, int *sign);
char *fcvt(double value, int ndigit, int *decpt, int *sign);
char *gcvt(double value, int ndigit, char *buf);

DESCRIPTION

The ecvt(), fcvt() and gcvt() functions convert floating-point numbers to null-terminated strings.

ecvt() Converts value to a null-terminated string of ndigit digits (where ndigit is reduced to an unspecified limit determined by the precision of a double) and returns a pointer to the string. The high-order digit is non-zero, unless the value is 0. The low-order digit is rounded. The position of the radix character relative to the beginning of the string is stored in the integer pointed to by decpt (negative means to the left of the returned digits). The radix character is not included in the returned string. If the sign of the result is negative, the integer pointed to by sign is non-zero, otherwise it is 0.

If the converted value is out of range or is not representable, the contents of the returned string are unspecified.

fcvt() Identical to ecvt() except that ndigit specifies the number of digits desired after the radix point. The total number of digits in the result string is restricted to an unspecified limit as determined by the precision of a double.

gcvt() Converts value to a null-terminated string (similar to that of the %g format of printf()) in the array pointed to by buf and returns buf. It produces ndigit significant digits (limited to an unspecified value determined by the precision of a double) in %f if possible, or %e (scientific notation) otherwise. A minus sign is included in the returned string if value is less than 0. A radix character is included in the returned string if value is not a whole number. Trailing zeros are suppressed where value is not a whole number. The radix character is determined by the current locale. If setlocale() has not been called successfully, the default locale, "POSIX", is used. The default locale specifies a period (.) as the radix character. The LC_NUMERIC category determines the value of the radix character within the current locale.

RETURN VALUE

The ecvt() and fcvt() functions return a pointer to a null-terminated string of digits.

The gcvt() function returns buf.

ERRORS

No errors are defined.

APPLICATION USAGE

The return values from ecvt() and fcvt() may point to static data which may be overwritten by subsequent calls to these functions.

For portability to implementations conforming to earlier versions of this document, sprintf() is preferred over this function.

SEE ALSO

printf(), setlocale(), <stdlib.h>.
CHANGE HISTORY
First released in Issue 4, Version 2.
encrypt()          CRYPT          System Interfaces

NAME
encrypt — encoding function

SYNOPSIS
EX
#include <unistd.h>

void encrypt (char block[64], int edflag);

DESCRIPTION
The encrypt() function provides (rather primitive) access to an implementation-dependent encoding algorithm. The key generated by setkey() is used to encrypt the string block with encrypt().

The block argument to encrypt() is an array of length 64 bytes containing only the bytes with numerical value of 0 and 1. The array is modified in place to a similar array using the key set by setkey(). If edflag is 0, the argument is encoded. If edflag is 1, the argument may be decoded (see the APPLICATION USAGE section below); if the argument is not decoded, errno will be set to [ENOSYS].

RETURN VALUE
The encrypt() function returns no value.

ERRORS
The encrypt() function will fail if:

[ENOSYS] The functionality is not supported on this implementation.

APPLICATION USAGE
In some environments, decoding may not be implemented. This is related to U.S. Government restrictions on encryption and decryption routines: the DES decryption algorithm cannot be exported outside the U.S.A. Historical practice has been to ship a different version of the encryption library without the decryption feature in the routines supplied. Thus the exported version of encrypt() does encoding but not decoding.

Because encrypt() does not return a value, applications wishing to check for errors should set errno to 0, call encrypt(), then test errno and, if it is non-zero, assume an error has occurred.

SEE ALSO
crypt(), setkey(), <unistd.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

• The header <unistd.h> is added to the SYNOPSIS section.

• The DESCRIPTION section is amended (a) to specify the encoding algorithm as implementation-dependent (b) to change “entry” to “function” and (c) to make decoding optional.

• The APPLICATION USAGE section is expanded to explain the restrictions on the availability of the DES decryption algorithm.

122 X/Open CAE Specification (1994)
NAME
endgrent, getgrent, setgrent — group database entry functions

SYNOPSIS
#include <grp.h>

void endgrent(void);
struct group *getgrent(void);
void setgrent(void);

DESCRIPTION
The getgrent() function returns a pointer to a structure containing the broken-out fields of an entry in the group database. When first called, getgrent() returns a pointer to a group structure containing the first entry in the group database. Thereafter, it returns a pointer to a group structure containing the next group structure in the group database, so successive calls may be used to search the entire database.

The setgrent() function effectively rewinds the group database to allow repeated searches.

The endgrent() function may be called to close the group database when processing is complete.

RETURN VALUE
When first called, getgrent() will return a pointer to the first group structure in the group database. Upon subsequent calls it returns the next group structure in the group database. The getgrent() function returns a null pointer on end-of-file or an error and errno may be set to indicate the error.

ERRORS
The getgrent() function may fail if:
[EINTR] A signal was caught during the operation.
[EIO] An I/O error has occurred.
[EMFILE] (OPEN_MAX) file descriptors are currently open in the calling process.
[ENFILE] The maximum allowable number of files is currently open in the system.

APPLICATION USAGE
The return value may point to a static area which is overwritten by a subsequent call to getgrgid(), getgrnam() or getgrent().

These functions are provided due to their historical usage. Applications should avoid dependencies on fields in the group database, whether the database is a single file, or where in the filesystem namespace the database resides. Applications should use getgrnam() and getgrgid() whenever possible both because it avoids these dependencies and for greater portability with systems that conform to earlier versions of this document.

SEE ALSO
getgrgid(), getgrnam(), getlogin(), getpwent(), <grp.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
endpwent( )

NAME
endpwent, getpwent, setpwent — user database functions

SYNOPSIS

```c
#include <pwd.h>

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

DESCRIPTION

The `getpwent()` function returns a pointer to a structure containing the broken-out fields of an entry in the user database. Each entry in the user database contains a `passwd` structure. When first called, `getpwent()` returns a pointer to a `passwd` structure containing the first entry in the user database. Thereafter, it returns a pointer to a `passwd` structure containing the next entry in the user database. Successive calls can be used to search the entire user database.

If an end-of-file or an error is encountered on reading, `getpwent()` returns a null pointer.

The `setpwent()` function effectively rewinds the user database to allow repeated searches.

The `endpwent()` function may be called to close the user database when processing is complete.

RETURN VALUE

The `getpwent()` function returns a null pointer on end-of-file or error.

ERRORS

The `getpwent()`, `setpwent()` and `endpwent()` functions may fail if:

- `[EIO]` An I/O error has occurred.

In addition, `getpwent()` and `setpwent()` may fail if:

- `[EMFILE]` `OPEN_MAX` file descriptors are currently open in the calling process.
- `[ENFILE]` The maximum allowable number of files is currently open in the system.

APPLICATION USAGE

The return value may point to a static area which is overwritten by a subsequent call to `getpwnam()`, `getpwuid()` or `getpwent()`.

These functions are provided due to their historical usage. Applications should avoid dependencies on fields in the password database, whether the database is a single file, or where in the filesystem namespace the database resides. Applications should use `getpwuid()` whenever possible both because it avoids these dependencies and for greater portability with systems that conform to earlier versions of this document.

SEE ALSO

`endgrent()`, `getlogin()`, `getpwnam()`, `getpwuid()`, `<pwd.h>`.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME

endutxent, getutxent, getutxid, getutxline, pututxline, setutxent — user accounting database functions

SYNOPSIS

UX

```
#include <utmpx.h>

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

DESCRIPTION

These functions provide access to the user accounting database.

The getutxent() function reads in the next entry from the user accounting database. If the database is not already open, it opens it. If it reaches the end of the database, it fails.

The getutxid() function searches forward from the current point in the database. If the ut_type value of the utmpx structure pointed to by id is BOOT_TIME, OLD_TIME or NEW_TIME, then it stops when it finds an entry with a matching ut_type value. If the ut_type value is INITPROCESS, LOGIN_PROCESS, USER_PROCESS, or DEAD_PROCESS, then it stops when it finds an entry whose type is one of these four and whose ut_id member matches the ut_id member of the utmpx structure pointed to by id. If the end of the database is reached without a match, getutxid() fails.

For all entries that match a request, the ut_type member indicates the type of the entry. Other members of the entry will contain meaningful data based on the value of the ut_type member as follows:

<table>
<thead>
<tr>
<th>ut_type Member</th>
<th>Other Members with Meaningful Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPTY</td>
<td>No others</td>
</tr>
<tr>
<td>BOOT_TIME</td>
<td>ut_tv</td>
</tr>
<tr>
<td>OLD_TIME</td>
<td>ut_tv</td>
</tr>
<tr>
<td>NEW_TIME</td>
<td>ut_tv</td>
</tr>
<tr>
<td>USER_PROCESS</td>
<td>ut_id, ut_user (login name of the user), ut_line, ut_pid, ut_tv</td>
</tr>
<tr>
<td>INIT_PROCESS</td>
<td>ut_id, ut_pid, ut_tv</td>
</tr>
<tr>
<td>LOGIN_PROCESS</td>
<td>ut_id, ut_user (implementation-specific name of the login process), ut_pid, ut_tv</td>
</tr>
<tr>
<td>DEAD_PROCESS</td>
<td>ut_id, ut_pid, ut_tv</td>
</tr>
</tbody>
</table>

The getutxline() function searches forward from the current point in the database until it finds an entry of the type LOGIN_PROCESS or USER_PROCESS which also has a ut_line value matching that in the utmpx structure pointed to by line. If the end of the database is reached without a match, getutxline() fails.

If the process has appropriate privileges, the pututxline() function writes out the structure into the user accounting database. It uses getutxid() to search for a record that satisfies the request. If this search succeeds, then the entry is replaced. Otherwise, a new entry is made at the end of the user accounting database.
The `setutxent()` function resets the input to the beginning of the database. This should be done before each search for a new entry if it is desired that the entire database be examined.

The `endutxent()` function closes the user accounting database.

**RETURN VALUE**

Upon successful completion, `getutxent()`, `getutxid()` and `getutxline()` return a pointer to a `utmpx` structure containing a copy of the requested entry in the user accounting database. Otherwise a null pointer is returned.

Upon successful completion, `pututxline()` returns a pointer to a `utmpx` structure containing a copy of the entry added to the user accounting database. Otherwise a null pointer is returned.

The `endutxent()` and `setutxent()` functions return no value.

**ERRORS**

No errors are defined for the `endutxent()`, `getutxent()`, `getutxid()`, `getutxline()` and `setutxent()` functions.

The `pututxline()` function may fail if:

- [EPERM] The process does not have appropriate privileges.

**APPLICATION USAGE**

The return value may point to a static area which is overwritten by a subsequent call to `getutxid()` or `getutxline()`.

Implementations may cache the data written by `getutxid()` or `getutxline()`. For this reason, to use `getutxline()` to search for multiple occurrences, it is necessary to zero out the static data after each success, or `getutxline()` could just return a pointer to the same `utmpx` structure over and over again.

There is one exception to the rule about removing the structure before further reads are done. The implicit read done by `pututxline()` (if it finds that it is not already at the correct place in the user accounting database) will not modify the static structure returned by `getutxent()`, `getutxid()` or `getutxline()`, if the application has just modified this structure and passed the pointer back to `pututxline()`.

The sizes of the arrays in the structure can be found using the `sizeof` operator.

**SEE ALSO**

- `<utmpx.h>`.

**CHANGE HISTORY**

NAME
   environ — array of character pointers to the environment strings

SYNOPSIS
   extern char **environ;

DESCRIPTION
   Refer to the XBD specification, Chapter 6, Environment Variables and exec.

APPLICATION USAGE
   The environ array should not be accessed directly by the application.

CHANGE HISTORY
   First released in Issue 1.
   Derived from Issue 1 of the SVID.
NAME
erand48 — generate uniformly distributed pseudo-random numbers

SYNOPSIS
#include <stdlib.h>

double erand48(unsigned short int xsubi[3]);

DESCRIPTION
Refer to drand48().

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:
• The <stdlib.h> header is added to the SYNOPSIS section.
NAME
erf, erfc — error and complementary error functions

SYNOPSIS
EX
#include <math.h>

double erf(double x);
double erfc(double x);

DESCRIPTION
The erf() function computes the error function of x, defined as:

\[ \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \]

The erfc() function computes \(1.0 - \text{erf}(x)\).

RETURN VALUE
Upon successful completion, erf() and erfc() return the value of the error function and complementary error function, respectively.

If x is NaN, NaN is returned and errno may be set to [EDOM].

If the correct value would cause underflow, 0 is returned and errno may be set to [ERANGE].

ERRORS
The erf() and erfc() functions may fail if:

[EDOM] The value of x is NaN.
[ERANGE] The result underflows.

No other errors will occur.

APPLICATION USAGE
The erfc() function is provided because of the extreme loss of relative accuracy if erf(x) is called for large x and the result subtracted from 1.0.

An application wishing to check for error situations should set errno to 0 before calling erf(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

SEE ALSO
isnan(), <math.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

- Removed references to matherr().
- The RETURN VALUE and ERRORS sections are substantially rewritten to rationalise error handling in the mathematics functions.
NAME
errno — XSI error return value

SYNOPSIS
#include <errno.h>

DESCRIPTION
The external variable or macro errno is used by many XSI functions to return error values. XSI-conformant systems may support the declaration:

    extern int errno;

Many functions provide an error number in errno which has type int and is defined in <errno.h>. The value of errno will be defined only after a call to a function for which it is explicitly stated to be set and until it is changed by the next function call. The value of errno should only be examined when it is indicated to be valid by a function’s return value. Programs should obtain the definition of errno by the inclusion of <errno.h>. The practice of defining errno in a program as extern int errno is obsolescent. No function in this specification sets errno to 0 to indicate an error.

It is unspecified whether errno is a macro or an identifier declared with external linkage. If a macro definition is suppressed in order to access an actual object, or a program defines an identifier with the name errno, the behaviour is undefined.

The value of errno is 0 at program startup, but is never set to 0 by any XSI function. The symbolic values stored in errno are documented in the ERRORS sections on all relevant pages.

APPLICATION USAGE
Previously both POSIX and X/Open documents were more restrictive than the ISO C standard in that they required errno to be defined as an external variable, whereas the ISO C standard required only that errno be defined as a modifiable lvalue with type int.

This revision is aligned with the ISO C standard; future versions of the ISO POSIX-1 standard are likely to require this more flexible definition. The historical usage is obsolescent; some implementations may not support it.

A program that uses errno for error checking should set it to 0 before a function call, then inspect it before a subsequent function call.

SEE ALSO
<errno.h>, Section 2.3 on page 25.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated for alignment with the ISO C standard:

- The DESCRIPTION section now guarantees that errno is set to 0 at program startup, and that it is never reset to 0 by any XSI function.

- The APPLICATION USAGE section is added. This revision is aligned with the ISO C standard, which permits errno to be a macro.

Another change is incorporated as follows:

- The FUTURE DIRECTIONS section is deleted.
NAME

environ, exec, execv, execl, execve, execlp, execvp — execute a file

SYNOPSIS

#include <unistd.h>
extern char **environ;

int execl(const char * path, const char * arg0, ... /*, (char *)0 */);
int execv(const char * path, char *const argv[]);
int execl(const char * path,
          const char *arg0, ... /*, (char *)0, char *const envp[]*/);
int execve(const char * path, char *const argv[], char *const envp[]);
int execlp(const char * file, const char * arg0, ... /*, (char *)0 */);
int execvp(const char * file, char *const argv[]);

DESCRIPTION

The exec functions replace the current process image with a new process image. The new image is constructed from a regular, executable file called the new process image file. There is no return from a successful exec, because the calling process image is overlaid by the new process image.

When a C-language program is executed as a result of this call, it is entered as a C-language function call as follows:

    int main (int argc, char *argv[]);

where argc is the argument count and argv is an array of character pointers to the arguments themselves. In addition, the following variable:

    extern char **environ;

is initialised as a pointer to an array of character pointers to the environment strings. The argv and environ arrays are each terminated by a null pointer. The null pointer terminating the argv array is not counted in argc.

The arguments specified by a program with one of the exec functions are passed on to the new process image in the corresponding main() arguments.

The argument path points to a pathname that identifies the new process image file.

The argument file is used to construct a pathname that identifies the new process image file. If the file argument contains a slash character, the file argument is used as the pathname for this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed as the environment variable PATH (see XBD specification, Chapter 6, Environment Variables). If this environment variable is not present, the results of the search are implementation-dependent.

If the process image file is not a valid executable object, execlp() and execvp() use the contents of that file as standard input to a command interpreter conforming to system(). In this case, the command interpreter becomes the new process image.

The arguments represented by arg0, ... are pointers to null-terminated character strings. These strings constitute the argument list available to the new process image. The list is terminated by a null pointer. The argument arg0 should point to a filename that is associated with the process being started by one of the exec functions.

The argument argv is an array of character pointers to null-terminated strings. The last member of this array must be a null pointer. These strings constitute the argument list available to the
new process image. The value in `argv[0]` should point to a filename that is associated with the process being started by one of the `exec` functions.

The argument `envp` is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process image. The `envp` array is terminated by a null pointer.

For those forms not containing an `envp` pointer (`execl()`, `execv()`, `execlp()` and `execvp()`), the environment for the new process image is taken from the external variable `environ` in the calling process.

The number of bytes available for the new process' combined argument and environment lists is `{ARG_MAX}`. It is implementation-dependent whether null terminators, pointers, and/or any alignment bytes are included in this total.

File descriptors open in the calling process image remain open in the new process image, except for those whose close-on-exec flag `FD_CLOEXEC` is set. For those file descriptors that remain open, all attributes of the open file description, including file locks remain unchanged.

Directory streams open in the calling process image are closed in the new process image.

The state of conversion descriptors and message catalogue descriptors in the new process image is undefined. For the new process, the equivalent of:

```c
setlocale(LC_ALL, "C")
```

is executed at startup.

Signals set to the default action (SIG_DFL) in the calling process image are set to the default action in the new process image. Signals set to be ignored (SIG_IGN) by the calling process image are set to be ignored by the new process image. Signals set to be caught by the calling process image are set to the default action in the new process image (see `<signal.h>`). After a successful call to any of the `exec` functions, alternate signal stacks are not preserved and the `SA_ONSTACK` flag is cleared for all signals.

After a successful call to any of the `exec` functions, any functions previously registered by `atexit()` are no longer registered.

If the `ST_NOSUID` bit is set for the file system containing the new process image file, then the effective user ID, effective group ID, saved set-user-ID and saved set-group-ID are unchanged in the new process image. Otherwise, if the set-user-ID mode bit of the new process image file is set, the effective user ID of the new process image is set to the user ID of the new process image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the effective group ID of the new process image is set to the group ID of the new process image file. The real user ID, real group ID, and supplementary group IDs of the new process image remain the same as those of the calling process image. The effective user ID and effective group ID of the new process image are saved (as the saved set-user-ID and the saved set-group-ID for use by `setuid()`).

Any shared memory segments attached to the calling process image will not be attached to the new process image.

Any mappings established through `mmap()` are not preserved across an `exec`.
The new process also inherits at least the following attributes from the calling process image:

- nice value (see `nice()`)
- `semadj` values (see `semop()`)
- process ID
- parent process ID
- process group ID
- session membership
- real user ID
- real group ID
- supplementary group IDs
- time left until an alarm clock signal (see `alarm()`)
- current working directory
- root directory
- file mode creation mask (see `umask()`)
- file size limit (see `ulimit()`)
- process signal mask (see `sigprocmask()`)
- pending signal (see `sigpending()`)
- `tms_utime`, `tms_stime`, `tms_cutime`, and `tms_cstime` (see `times()`)
- resource limits
- controlling terminal
- interval timers

All other process attributes defined in this document will be the same in the new and old process images. The inheritance of process attributes not defined by this document is implementation-dependent.

Upon successful completion, the `exec` functions mark for update the `st_atime` field of the file. If an `exec` function failed but was able to locate the process image file, whether the `st_atime` field is marked for update is unspecified. Should the `exec` function succeed, the process image file is considered to have been opened with `open()`. The corresponding `close()` is considered to occur at a time after this open, but before process termination or successful completion of a subsequent call to one of the `exec` functions. The `argv[]` and `envp[]` arrays of pointers and the strings to which those arrays point will not be modified by a call to one of the `exec` functions, except as a consequence of replacing the process image.

**RETURN VALUE**

If one of the `exec` functions returns to the calling process image, an error has occurred; the return value is −1, and `errno` is set to indicate the error.

**ERRORS**

The `exec` functions will fail if:

- [E2BIG] The number of bytes used by the new process image’s argument list and environment list is greater than the system-imposed limit of `{ARG_MAX}` bytes.
- [EACCES] Search permission is denied for a directory listed in the new process image file’s path prefix, or the new process image file denies execution permission, or the new process image file is not a regular file and the implementation does not support execution of files of its type.
- [ELOOP] Too many symbolic links were encountered in resolving `path`.
- [ENAMETOOLONG] The length of the `path` or `file` arguments, or an element of the environment variable `PATH` prefixed to a file, exceeds `{PATH_MAX}`, or a pathname
component is longer than \{NAME_MAX\}.

- **[ENOENT]**: A component of `path` or `file` does not name an existing file or `path` or `file` is an empty string.
- **[ENOTDIR]**: A component of the new process image file's path prefix is not a directory.

The `exec` functions, except for `execlp()` and `execvp()`, will fail if:

- **[ENOEXEC]**: The new process image file has the appropriate access permission but is not in the proper format.

The `exec` functions may fail if:

**UX**

- **[ENAMETOOLONG]**: Pathname resolution of a symbolic link produced an intermediate result whose length exceeds \{PATH_MAX\}.

- **[ENOMEM]**: The new process image requires more memory than is allowed by the hardware or system-imposed memory management constraints.

**EX**

- **[ETXTBSY]**: The new process image file is a pure procedure (shared text) file that is currently open for writing by some process.

**APPLICATION USAGE**

As the state of conversion descriptors and message catalogue descriptors in the new process image is undefined, portable applications should not rely on their use and should close them prior to calling one of the `exec` functions.

Applications that require other than the default POSIX locale should call `setlocale()` with the appropriate parameters to establish the locale of the new process.

**SEE ALSO**

`alarm()`, `atexit()`, `chmod()`, `exit()`, `fcntl()`, `fork()`, `fstatvfs()`, `getenv()`, `getitimer()`, `getrlimit()`, `mmap()`, `nice()`, `putenv()`, `semop()`, `setlocale()`, `shmat()`, `sigaction()`, `sigaltstack()`, `sigpending()`, `sigprocmask()`, `system()`, `times()`, `ulimit()`, `umask()`, `<unistd.h>`, XBD specification, Chapter 9, General Terminal Interface.

**CHANGE HISTORY**

First released in Issue 1.

Derived from Issue 1 of the SVID.

**Issue 4**

The following change is incorporated for alignment with the ISO POSIX-1 standard:

- In the **ERRORS** section, (a) the description of the `[ENOEXEC]` error is changed to indicate that this error does not apply to `execlp()` and `execvp()`, and (b) the `[ENOMEM]` error is added.

The following change is incorporated for alignment with the FIPS requirements:

- In the **ERRORS** section, the condition whereby `[ENAMETOOLONG]` will be returned if a pathname component is larger that `NAME_MAX` is now defined as mandatory and marked as an extension.

Other changes are incorporated as follows:

- The header `<unistd.h>` is added to the **SYNOPSIS** section.

- The **const** keyword is added to identifiers of constant type (for example, `path`, `file`).

- In the **DESCRIPTION** section, (a) an indication of the disposition of conversion descriptors after a call to one of the `exec` functions is added, (b) a statement about the interaction between
exec and atexit() is added, (c) “usually” in the descriptions of argument pointers is removed, (d) “owner ID” is changed to “user ID”, (e) shared memory is no longer optional and (f) the penultimate paragraph is changed to correct an error in Issue 3: it now refers to “All other process attributes...” instead of “All the process attributes....”

- A note about the initialisation of locales is added to the APPLICATION USAGE section.

### Issue 4, Version 2

The following changes are incorporated for X/OPEN UNIX conformance:

- The DESCRIPTION is changed to indicate the disposition of alternate signal stacks, the SA_ONSTACK flag and mappings established through mmap() after a successful call to one of the exec functions. The effects of ST_NOSUID being set for a file system are defined. A statement is added that mappings established through mmap() are not preserved across an exec. The list of inherited process attributes is extended to include resource limits, the controlling terminal and interval timers.

- In the ERRORS section, the condition whereby [ELOOP] will be returned if too many symbolic links are encountered during pathname resolution is defined as mandatory.

- In the ERRORS section, a second [ENAMETOOLONG] condition is defined that may report excessive length of an intermediate result of pathname resolution of a symbolic link.
NAME
exit, _exit — terminate process

SYNOPSIS
#include <stdlib.h>
void exit(int status);
#include <unistd.h>
void _exit(int status);

DESCRIPTION
The exit() function first calls all functions registered by atexit(), in the reverse order of their registration. Each function is called as many times as it was registered.

If a function registered by a call to atexit() fails to return, the remaining registered functions are not called and the rest of the exit() processing is not completed. If exit() is called more than once, the effects are undefined.

The exit() function then flushes all output streams, closes all open streams, and removes all files created by tmpfile().

The _exit() and exit() functions terminate the calling process with the following consequences:

- All of the file descriptors, directory streams, conversion descriptors and message catalogue descriptors opened in the calling process are closed.
- If the parent process of the calling process is executing a wait(), wait3(), waitid() or waitpid(), and has neither set its SA_NOCLDWAIT flag nor set SIGCHLD to SIG_IGN, it is notified of the calling process' termination and the low-order eight bits (that is, bits 0377) of status are made available to it. If the parent is not waiting, the child's status will be made available to it when the parent subsequently executes wait(), wait3(), waitid() or waitpid().
- If the parent process of the calling process is not executing a wait(), wait3(), waitid() or waitpid(), and has not set its SA_NOCLDWAIT flag, nor set SIGCHLD to SIG_IGN, the calling process is transformed into a zombie process. A zombie process is an inactive process and it will be deleted at some later time when its parent process executes wait(), wait3(), waitid() or waitpid().
- Termination of a process does not directly terminate its children. The sending of a SIGHUP signal as described below indirectly terminates children in some circumstances.
- If the implementation supports the SIGCHLD signal, a SIGCHLD will be sent to the parent process.
- The parent process ID of all of the calling process' existing child processes and zombie processes is set to the process ID of an implementation-dependent system process. That is, these processes are inherited by a special system process.
- Each mapped memory object is unmapped.
- Each attached shared-memory segment is detached and the value of shm_nattch (see shmget()) in the data structure associated with its shared memory ID is decremented by 1.
- For each semaphore for which the calling process has set a semadj value, see semop(), that value is added to the semval of the specified semaphore.
- If the process is a controlling process, the SIGHUP signal will be sent to each process in the foreground process group of the controlling terminal belonging to the calling process.
If the process is a controlling process, the controlling terminal associated with the session is disassociated from the session, allowing it to be acquired by a new controlling process.

If the exit of the process causes a process group to become orphaned, and if any member of the newly-orphaned process group is stopped, then a SIGHUP signal followed by a SIGCONT signal will be sent to each process in the newly-orphaned process group.

**RETURN VALUE**
These functions do not return.

**APPLICATION USAGE**
 Normally applications should use `exit()` rather than `_exit()`.

**ERRORS**
No errors are defined.

**SEE ALSO**
`atexit()`, `close()`, `fclose()`, `semop()`, `shmget()`, `sigaction()`, `wait()`, `wait3()`, `waitid()`, `waitpid()`, `<stdlib.h>`, `<unistd.h>`.

**CHANGE HISTORY**
First released in Issue 1.

Derived from Issue 1 of the SVID.

**Issue 4**
The following change is incorporated for alignment with the ISO C standard:

- In the DESCRIPTION section, (a) interactions between `exit()` and `atexit()` are defined, and (b) it is now stated explicitly that all files created by `tmpfile()` are removed.

Other changes are incorporated as follows:

- The header `<unistd.h>` is added to the SYNOPSIS for `_exit()`.
- In the DESCRIPTION, text describing (a) the behaviour when a function registered by `atexit()` fails to return, and (b) consequences of calling `exit()` more than once, are added.
- The phrase ‘If the implementation supports job control’ is removed from the last bullet in the DESCRIPTION section. This is because job control is now defined as mandatory for all conforming implementations.

**Issue 4, Version 2**
The following changes to the DESCRIPTION are incorporated for X/OPEN UNIX conformance:

- References to the functions `wait3()` and `waitid()` are added in appropriate places throughout the text.
- Interactions with the SA_NOCLDWAIT flag and SIGCHLD signal are defined.
- It is specified that each mapped memory object is unmapped.
exp()  

NAME  
exp — exponential function  

SYNOPSIS  
#include <math.h>  
double exp(double x);  

DESCRIPTION  
The exp() function computes the exponent of x, defined as \( e^x \).  

RETURN VALUE  
Upon successful completion, exp() returns the exponential value of x.  
If the correct value would cause overflow, exp() returns HUGE_VAL and sets errno to [ERANGE].  
If the correct value would cause underflow, exp() returns 0 and may set errno to [ERANGE].  
EX  
If x is NaN, NaN is returned and errno may be set to [EDOM].  

ERRORS  
The exp() function will fail if:  
[ERANGE] The result overflows.  
The exp() function may fail if:  
EX [EDOM] The value of x is NaN.  
[ERANGE] The result underflows.  
EX [EDOM] No other errors will occur.  

APPLICATION USAGE  
An application wishing to check for error situations should set errno to 0 before calling exp(). If errno is non-zero on return, or the return value is NaN, an error has occurred.  

SEE ALSO  
isnan(), log(), <math.h>.  

CHANGE HISTORY  
First released in Issue 1.  
Derived from Issue 1 of the SVID.  

Issue 4  
The following changes are incorporated in this issue:  
• Removed references to matherr().  
• The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the ISO C standard and to rationalise error handling in the mathematics functions.  
• The return value specified for [EDOM] is marked as an extension.
NAME
expm1 — computes exponential functions

SYNOPSIS
#include <math.h>

double expm1 (double x);

DESCRIPTION
The expm1() function computes $e^x - 1.0$.

RETURN VALUE
If $x$ is NaN, then the function returns NaN and errno may be set to EDOM.
If $x$ is positive infinity, expm1() returns positive infinity.
If $x$ is negative infinity, expm1() returns −1.0.
If the value overflows, expm1() returns HUGE_VAL and may set errno to ERANGE.

ERRORS
The expm1() function may fail if:
[EDOM] The value of $x$ is NaN.
[ERANGE] The result overflows.

APPLICATION USAGE
The value of expm1($x$) may be more accurate than exp($x$)−1.0 for small values of $x$.
The expm1() and log1p() functions are useful for financial calculations of $((1+x)^n−1)/x$, namely:
expm1($n \times log1p(x))/x$
when $x$ is very small (for example, when calculating small daily interest rates). These functions also simplify writing accurate inverse hyperbolic functions.

SEE ALSO
exp(), ilogb(), log1p(), <math.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
fabs() — absolute value function

SYNOPSIS

#include <math.h>

double fabs(double x);

DESCRIPTION

The fabs() function computes the absolute value of $x$, $|x|$.

RETURN VALUE

Upon successful completion, fabs() returns the absolute value of $x$.

EX

If $x$ is NaN, NaN is returned and errno may be set to [EDOM]. If the result underflows, 0 is returned and errno may be set to [ERANGE].

ERRORS

The fabs() function may fail if:

EX

[EDOM] The value of $x$ is NaN.

[ERANGE] The result underflows.

EX

No other errors will occur.

APPLICATION USAGE

An application wishing to check for error situations should set errno to 0 before calling fabs(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

SEE ALSO

isnan(), <math.h>.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated in this issue:

- Removed references to matherr().
- The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the ISO C standard and to rationalise error handling in the mathematics functions.
- The return value specified for [EDOM] is marked as an extension.
NAME
fattach — attach a STREAMS-based file descriptor to a file in the file system name space

SYNOPSIS

#include <stropts.h>

int fattach(int fildes, const char *path);

DESCRIPTION

The fattach() function attaches a STREAMS-based file descriptor to a file, effectively associating
a pathname with fildes. The fildes argument must be a valid open file descriptor associated with
a STREAMS file. The path argument points to a pathname of an existing file. The process must
have appropriate privileges, or must be the owner of the file named by path and have write
permission. A successful call to fattach() causes all pathnames that name the file named by path
to name the STREAMS file associated with fildes, until the STREAMS file is detached from the
file. A STREAMS file can be attached to more than one file and can have several pathnames
associated with it.

The attributes of the named STREAMS file are initialised as follows: the permissions, user ID,
group ID, and times are set to those of the file named by path, the number of links is set to 1, and
the size and device identifier are set to those of the STREAMS file associated with fildes. If any
attributes of the named STREAMS file are subsequently changed (for example, by chmod()),
neither the attributes of the underlying file nor the attributes of the STREAMS file to which fildes
refers are affected.

File descriptors referring to the underlying file, opened prior to an fattach() call, continue to refer
to the underlying file.

RETURN VALUE

Upon successful completion, fattach() returns 0. Otherwise, −1 is returned and errno is set to
indicate the error.

ERRORS

The fattach() function will fail if:

[EACCES] Search permission is denied for a component of the path prefix, or the process
is the owner of path but does not have write permissions on the file named by
path.

[EBADF] The fildes argument is not a valid open file descriptor.

[ENOENT] A component of path does not name an existing file or path is an empty string.

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

[EPERM] The effective user ID of the process is not the owner of the file named by path
and the process does not have appropriate privilege.

[EBUSY] The file named by path is currently a mount point or has a STREAMS file
attached to it.

[ENAMETOOLONG] The size of path exceeds [PATH_MAX], or a component of path is longer than
(NAME_MAX).

[ELOOP] Too many symbolic links were encountered in resolving path.

The fattach() function may fail if:

[EINVAL] The fildes argument does not refer to a STREAMS file.
[ENAMETOOLONG]
Pathname resolution of a symbolic link produced an intermediate result whose length exceeds [PATH_MAX].

SEE ALSO
fdetach(), isastream(), <stropts.h>.

APPLICATION USAGE
The fattach() function behaves similarly to the traditional mount() function in the way a file is temporarily replaced by the root directory of the mounted file system. In the case of fattach(), the replaced file need not be a directory and the replacing file is a STREAMS file.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
fchdir — change working directory

SYNOPSIS

```c
#include <unistd.h>

int fchdir(int fildes);
```

DESCRIPTION
The `fchdir()` function has the same effect as `chdir()` except that the directory that is to be the new current working directory is specified by the file descriptor `fildes`.

RETURN VALUE
Upon successful completion, `fchdir()` returns 0. Otherwise, it returns −1 and sets `errno` to indicate the error. On failure the current working directory remains unchanged.

ERRORS
The `fchdir()` function will fail if:

- `[EACCES]` Search permission is denied for the directory referenced by `fildes`.
- `[EBADF]` The `fildes` argument is not an open file descriptor.
- `[ENOTDIR]` The open file descriptor `fildes` does not refer to a directory.

The `fchdir()` may fail if:

- `[EINTR]` A signal was caught during the execution of `fchdir()`.
- `[EIO]` An I/O error occurred while reading from or writing to the file system.

SEE ALSO
`chdir()`, `chroot()`, `<unistd.h>`.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
fchmod — change mode of file

SYNOPSIS
UX
#include <sys/stat.h>

int fchmod(int fildes, mode_t mode);

DESCRIPTION
The fchmod() function has the same effect as chmod() except that the file whose permissions are to be changed is specified by the file descriptor fildes.

RETURN VALUE
Upon successful completion, fchmod() returns 0. Otherwise, it returns −1 and sets errno to indicate the error.

ERRORS
The fchmod() function will fail if:

[EBADF]  The fildes argument is not an open file descriptor.
[EROFS]  The file referred to by fildes resides on a read-only file system.
[EPERM]  The effective user ID does not match the owner of the file and the process does not have appropriate privilege.

The fchmod() function may fail if:

[EINTR]  The fchmod() function was interrupted by a signal.
[EINVAL]  The value of the mode argument is invalid.

SEE ALSO
chmod(), chown(), creat(), fcntl(), fstatvfs(), mknod(), open(), read(), stat(), write(), <sys/stat.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
fchown — change owner and group of a file

SYNOPSIS

```c
#include <unistd.h>

int fchown(int fildes, uid_t owner, gid_t group);
```

DESCRIPTION
The `fchown()` function has the same effect as `chown()` except that the file whose owner and group are to be changed is specified by the file descriptor `fildes`.

RETURN VALUE
Upon successful completion, `fchown()` returns 0. Otherwise, it returns -1 and sets `errno` to indicate the error.

ERRORS
The `fchown()` function will fail if:

- **[EBADF]** The `fildes` argument is not an open file descriptor.
- **[EPERM]** The effective user ID does not match the owner of the file or the process does not have appropriate privilege.
- **[EROFS]** The file referred to by `fildes` resides on a read-only file system.

The `fchown()` function may fail if:

- **[EINVAL]** The owner or group ID is not a value supported by the implementation.
- **[EIO]** A physical I/O error has occurred.
- **[EINTR]** The `fchown()` function was interrupted by a signal which was caught.

SEE ALSO
`chown()`, `<unistd.h>`.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
fclose — close a stream

SYNOPSIS
#include <stdio.h>

int fclose(FILE * stream);

DESCRIPTION
The fclose() function causes the stream pointed to by stream to be flushed and the associated file to be closed. Any unwritten buffered data for the stream is written to the file; any unread buffered data is discarded. The stream is disassociated from the file. If the associated buffer was automatically allocated, it is deallocated. It marks for update the st_ctime and st_mtime fields of the underlying file, if the stream was writable, and if buffered data had not been written to the file yet. The fclose() function will perform a close() on the file descriptor that is associated with the stream pointed to by stream.

After the call to fclose(), any use of stream causes undefined behaviour.

RETURN VALUE
Upon successful completion, fclose() returns 0. Otherwise, it returns EOF and sets errno to indicate the error.

ERRORS
The fclose() function will fail if:

[EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying stream and the process would be delayed in the write operation.

[EBADF] The file descriptor underlying stream is not valid.

[EBIG] An attempt was made to write a file that exceeds the maximum file size or the process’ file size limit.

[EINTR] The fclose() function was interrupted by a signal.

[EIO] The process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU and the process group of the process is orphaned. This error may also be returned under implementation-dependent conditions.

[ENOSPC] There was no free space remaining on the device containing the file.

[EPipe] An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal will also be sent to the process.

The fclose() function may fail if:

[ENXIO] A request was made of a non-existent device, or the request was outside the device.

SEE ALSO
close(), fopen(), getrlimit(), ulimit(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.
Issue 4

The following changes are incorporated in this issue:

• The last sentence of the first paragraph in the DESCRIPTION section is changed to say close() instead of fclose(). This was an error in Issue 3.

• The following paragraph is withdrawn from the DESCRIPTION section (by POSIX as well as X/Open) because of the possibility of causing applications to malfunction, and the impossibility of implementing these mechanisms for pipes:

   If the file is not already at EOF, and the file is one capable of seeking, the file offset of the underlying open file description will be adjusted so that the next operation on the open file description deals with the byte after the last one read from or written to the stream being closed.

   It is replaced with a statement that any subsequent use of stream is undefined.

• The [EFBIG] error is marked to indicate the extensions.

Issue 4, Version 2

A cross-reference to getrlimit() is added.
**NAME**

fcntl — file control

**SYNOPSIS**

```c
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>

int fcntl(int fildes, int cmd, ...);
```

**DESCRIPTION**

The `fcntl()` function provides for control over open files. The `fildes` argument is a file descriptor.

The available values for `cmd` are defined in the header `<fcntl.h>`, which include:

- **F_DUPFD**
  Return a new file descriptor which is the lowest numbered available (that is, not already open) file descriptor greater than or equal to the third argument, `arg`, taken as an integer of type `int`. The new file descriptor refers to the same open file description as the original file descriptor, and shares any locks. The FD_CLOEXEC flag associated with the new file descriptor is cleared to keep the file open across calls to one of the `exec` functions.

- **F_GETFD**
  Get the file descriptor flags defined in `<fcntl.h>` that are associated with the file descriptor `fildes`. File descriptor flags are associated with a single file descriptor and do not affect other file descriptors that refer to the same file.

- **F_SETFD**
  Set the file descriptor flags defined in `<fcntl.h>`, that are associated with `fildes`, to the third argument, `arg`, taken as type `int`. If the FD_CLOEXEC flag in the third argument is 0, the file will remain open across the `exec` functions; otherwise the file will be closed upon successful execution of one of the `exec` functions.

- **F_GETFL**
  Get the file status flags and file access modes, defined in `<fcntl.h>`, for the file description associated with `fildes`. The file access modes can be extracted from the return value using the mask O_ACCMODE, which is defined in `<fcntl.h>`. File status flags and file access modes are associated with the file description and do not affect other file descriptors that refer to the same file with different open file descriptions.

- **F_SETFL**
  Set the file status flags, defined in `<fcntl.h>`, for the file description associated with `fildes` from the corresponding bits in the third argument, `arg`, taken as type `int`. Bits corresponding to the file access mode and the `oflag` values that are set in `arg` are ignored. If any bits in `arg` other than those mentioned here are changed by the application, the result is unspecified.

The following commands are available for advisory record locking. Record locking is supported for regular files, and may be supported for other files.

- **F_GETLK**
  Get the first lock which blocks the lock description pointed to by the third argument, `arg`, taken as a pointer to type `struct flock`, defined in `<fcntl.h>`. The information retrieved overwrites the information passed to `fcntl()` in the structure `flock`. If no lock is found that would prevent this lock from being created, then the structure will be left unchanged except for the lock type which will be set to F_UNLCK.
F_SETLK
Set or clear a file segment lock according to the lock description pointed to by the third argument, \textit{arg}, taken as a pointer to type \textit{struct flock}, defined in <\texttt{fcntl.h}>. F_SETLK is used to establish shared (or read) locks (F_RDLCK) or exclusive (or write) locks (F_WRLCK), as well as to remove either type of lock (F_UNLCK). F_RDLCK, F_WRLCK and F_UNLCK are defined in <\texttt{fcntl.h}>. If a shared or exclusive lock cannot be set, \texttt{fcntl()} will return immediately with a return value of $-1$.

F_SETLKW
This command is the same as F_SETLK except that if a shared or exclusive lock is blocked by other locks, the process will wait until the request can be satisfied. If a signal that is to be caught is received while \texttt{fcntl()} is waiting for a region, \texttt{fcntl()} will be interrupted. Upon return from the process’ signal handler, \texttt{fcntl()} will return $-1$ with \texttt{errno} set to [EINTR], and the lock operation will not be done.

Additional implementation-dependent commands may be defined in <\texttt{fcntl.h}>. Their names will start with F_.

When a shared lock is set on a segment of a file, other processes will be able to set shared locks on that segment or a portion of it. A shared lock prevents any other process from setting an exclusive lock on any portion of the protected area. A request for a shared lock will fail if the file descriptor was not opened with read access.

An exclusive lock will prevent any other process from setting a shared lock or an exclusive lock on any portion of the protected area. A request for an exclusive lock will fail if the file descriptor was not opened with write access.

The structure \texttt{flock} describes the type (\texttt{l_type}), starting offset (\texttt{l_whence}), relative offset (\texttt{l_start}), size (\texttt{l_len}) and process ID (\texttt{l_pid}) of the segment of the file to be affected.

The value of \texttt{l_whence} is SEEK_SET, SEEK_CUR or SEEK_END, to indicate that the relative offset \texttt{l_start} bytes will be measured from the start of the file, current position or end of the file, respectively. The value of \texttt{l_len} is the number of consecutive bytes to be locked. The value of \texttt{l_len} may be negative (where the definition of \texttt{off_t} permits negative values of \texttt{l_len}). The \texttt{l_pid} field is only used with F_GETLK to return the process ID of the process holding a blocking lock. After a successful F_GETLK request, that is, one in which a lock was found, the value of \texttt{l_whence} will be SEEK_SET.

If \texttt{l_len} is positive, the area affected starts at \texttt{l_start} and ends at \texttt{l_start} + \texttt{l_len}−1. If \texttt{l_len} is negative, the area affected starts at \texttt{l_start} + \texttt{l_len} and ends at \texttt{l_start}−1. Locks may start and extend beyond the current end of a file, but must not be negative relative to the beginning of the file. A lock will be set to extend to the largest possible value of the file offset for that file by setting \texttt{l_len} to 0. If such a lock also has \texttt{l_start} set to 0 and \texttt{l_whence} is set to SEEK_SET, the whole file will be locked.

There will be at most one type of lock set for each byte in the file. Before a successful return from an F_SETLK or an F_SETLKW request when the calling process has previously existing locks on bytes in the region specified by the request, the previous lock type for each byte in the specified region will be replaced by the new lock type. As specified above under the descriptions of shared locks and exclusive locks, an F_SETLK or an F_SETLKW request will (respectively) fail or block when another process has existing locks on bytes in the specified region and the type of any of those locks conflicts with the type specified in the request.

All locks associated with a file for a given process are removed when a file descriptor for that file is closed by that process or the process holding that file descriptor terminates. Locks are not inherited by a child process created using \texttt{fork()}. 

EX
A potential for deadlock occurs if a process controlling a locked region is put to sleep by attempting to lock another process’ locked region. If the system detects that sleeping until a locked region is unlocked would cause a deadlock, `fcntl()` will fail with an [EDEADLK] error.

RETURN VALUE

Upon successful completion, the value returned depends on `cmd` as follows:

- **F_DUPFD**: A new file descriptor.
- **F_GETFD**: Value of flags defined in `<fcntl.h>`. The return value will not be negative.
- **F_SETFD**: Value other than −1.
- **F_GETFL**: Value of file status flags and access modes. The return value will not be negative.
- **F_SETFL**: Value other than −1.
- **F_GETLK**: Value other than −1.
- **F_SETLK**: Value other than −1.
- **F_SETLKW**: Value other than −1.

Otherwise, −1 is returned and `errno` is set to indicate the error.

ERRORS

The `fcntl()` function will fail if:

- **[EACCES]** or **[EAGAIN]**: The `cmd` argument is `F_SETLK`; the type of lock (`l_type`) is a shared (F_RDLCK) or exclusive (F_WRLCK) lock and the segment of a file to be locked is already exclusive-locked by another process, or the type is an exclusive lock and some portion of the segment of a file to be locked is already shared-locked or exclusive-locked by another process.

- **[EBADF]**: The `fildes` argument is not a valid open file descriptor, or the argument `cmd` is `F_SETLK` or `F_SETLKW`, the type of lock, `l_type`, is a shared lock (F_RDLCK), and `fildes` is not a valid file descriptor open for reading, or the type of lock `l_type`, is an exclusive lock (F_WRLCK), and `fildes` is not a valid file descriptor open for writing.

- **[EINTR]**: The `cmd` argument is `F_SETLKW` and the function was interrupted by a signal.

- **[EINVAL]**: The `cmd` argument is invalid, or the `cmd` argument is `F_DUPFD` and `arg` is negative or greater than or equal to `{OPEN_MAX}`, or the `cmd` argument is `F_GETLK`, `F_SETLK` or `F_SETLKW` and the data pointed to by `arg` is not valid, or `fildes` refers to a file that does not support locking.

- **[EMFILE]**: The argument `cmd` is `F_DUPFD` and `{OPEN_MAX}` file descriptors are currently open in the calling process, or no file descriptors greater than or equal to `arg` are available.

- **[ENOLCK]**: The argument `cmd` is `F_SETLK` or `F_SETLKW` and satisfying the lock or unlock request would result in the number of locked regions in the system exceeding a system-imposed limit.
The `fcntl()` function may fail if:

```
[EDEADLK] The cmd argument is F_SETLKW, the lock is blocked by some lock from another process and putting the calling process to sleep, waiting for that lock to become free would cause a deadlock.
```

**SEE ALSO**

`close()`, `exec`, `open()`, `sigaction()`, `<fcntl.h>`, `<signal.h>`, `<sys/types.h>`, `<unistd.h>`.

**CHANGE HISTORY**

First released in Issue 1.

Derived from Issue 1 of the SVID.

**Issue 4**

The following change is incorporated for alignment with the ISO POSIX-1 standard:

- In the **DESCRIPTION** section, the meaning of a successful F_SETLK or F_SETLKW request is clarified, after a POSIX Request for Interpretation.

Other changes are incorporated as follows:

- The headers `<sys/types.h>` and `<unistd.h>` are now marked as optional (OH); these headers do not need to be included on XSI-conformant systems.

- In the **DESCRIPTION** section (a) sentences describing behaviour when `l_len` is negative are marked as an extension and (b) the description of locks is corrected to make it a requirement on the application.
NAME

fcvt — convert floating-point number to string

SYNOPSIS

```c
#include <stdlib.h>

char *fcvt(double value, int ndigit, int *decpt, int *sign);
```

DESCRIPTION

Refer to `ecvt()`.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
FD_CLR — macros for synchronous I/O multiplexing

SYNOPSIS

```c
#include <sys/time.h>

FD_CLR(int fd, fd_set *fdset);
FD_ISSET(int fd, fd_set *fdset);
FD_SET(int fd, fd_set *fdset);
FD_ZERO(fd_set *fdset);
```

DESCRIPTION
Refer to `select()`.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
fdetach — detach a name from a STREAMS-based file descriptor

SYNOPSIS
UX
#include <stropts.h>
int fdetach(const char *path);

DESCRIPTION
The fdetach() function detaches a STREAMS-based file from the file to which it was attached by a previous call to fattach(). The path argument points to the pathname of the attached STREAMS file. The process must have appropriate privileges or be the owner of the file. A successful call to fdetach() causes all pathnames that named the attached STREAMS file to again name the file to which the STREAMS file was attached. All subsequent operations on path will operate on the underlying file and not on the STREAMS file.

All open file descriptions established while the STREAMS file was attached to the file referenced by path, will still refer to the STREAMS file after the fdetach() has taken effect.

If there are no open file descriptors or other references to the STREAMS file, then a successful call to fdetach() has the same effect as performing the last close() on the attached file.

RETURN VALUE
Upon successful completion, fdetach() returns 0. Otherwise, it returns −1 and sets errno to indicate the error.

ERRORS
The fdetach() function will fail if:

[EACCES] Search permission is denied on a component of the path prefix.
[EPERM] The effective user ID is not the owner of path and the process does not have appropriate privileges.
[ENOTDIR] A component of the path prefix is not a directory.
[ENOENT] A component of path does not name an existing file or path is an empty string.
[EINVAL] The path argument names a file that is not currently attached.
[ENAMETOOLONG] The size of a pathname exceeds {PATH_MAX}, or a pathname component is longer than {NAME_MAX}.
[ELOOP] Too many symbolic links were encountered in resolving path.

The fdetach() function may fail if:

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

SEE ALSO
fattach(), <stropts.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
fdopen — associate a stream with a file descriptor

SYNOPSIS
#include <stdio.h>
FILE *fdopen(int fildes, const char * mode);

DESCRIPTION
The `fdopen()` function associates a stream with a file descriptor.

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

- `r` or `rb` open a file for reading
- `w` or `wb` open a file for writing
- `a` or `ab` open a file for writing at end of file
- `r+` or `rb+` or `r+b` open a file for update (reading and writing)
- `w+` or `wb+` or `w+b` open a file for update (reading and writing)
- `a+` or `ab+` or `a+b` open a file for update (reading and writing) at end of file

The meaning of these flags is exactly as specified in `fopen()`, except that modes beginning with `w` do not cause truncation of the file.

Additional values for the `mode` argument may be supported by an implementation.

The mode of the stream must be allowed by the file access mode of the open file. The file position indicator associated with the new stream is set to the position indicated by the file offset associated with the file descriptor.

The error and end-of-file indicators for the stream are cleared. The `fdopen()` function may cause the `st_atime` field of the underlying file to be marked for update.

RETURN VALUE
Upon successful completion, `fdopen()` returns a pointer to a stream. Otherwise, a null pointer is returned and `errno` is set to indicate the error.

ERRORS
The `fdopen()` function may fail if:

- `[EBADF]` The `fildes` argument is not a valid file descriptor.
- `[EINVAL]` The `mode` argument is not a valid mode.
- `[EMFILE]` [FOPEN_MAX] streams are currently open in the calling process.
- `[EMFILE]` [STREAM_MAX] streams are currently open in the calling process.
- `[ENOMEM]` Insufficient space to allocate a buffer.

APPLICATION USAGE
{STREAM_MAX} is the number of streams that one process can have open at one time. If defined, it has the same value as [FOPEN_MAX].

File descriptors are obtained from calls like `open()`, `dup()`, `creat()` or `pipe()`, which open files but do not return streams.

SEE ALSO
`fclose()`, `fopen()`, `open()`, `<stdio.h>`, Section 2.4.1 on page 32.
CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:
• The type of argument mode is changed from char * to const char *.

Other changes are incorporated as follows:
• In the DESCRIPTION section, the use and settings of the mode argument are changed to include binary streams and are marked as extensions.
• All errors identified in the ERRORS section are marked as extensions, and the [EMFILE] error is added.
• The APPLICATION USAGE section is added.
NAME
feof — test end-of-file indicator on a stream

SYNOPSIS
#include <stdio.h>
int feof(FILE *stream);

DESCRIPTION
The feof() function tests the end-of-file indicator for the stream pointed to by stream.

RETURN VALUE
The feof() function returns non-zero if and only if the end-of-file indicator is set for stream.

ERRORS
No errors are defined.

SEE ALSO
clearerr(), ferror(), fopen(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:
• The ERRORS section is rewritten, such that no error return values are now defined for this interface.
ferror()                                BASE                                System Interfaces

NAME
  ferror — test error indicator on a stream

SYNOPSIS
  #include <stdio.h>
  int ferror(FILE *stream);

DESCRIPTION
  The ferror() function tests the error indicator for the stream pointed to by stream.

RETURN VALUE
  The ferror() function returns non-zero if and only if the error indicator is set for stream.

ERRORS
  No errors are defined.

SEE ALSO
  clearerr(), feof(), fopen(), <stdio.h>.

CHANGE HISTORY
  First released in Issue 1.
  Derived from Issue 1 of the SVID.

Issue 4
  The following change is incorporated in this issue:
    • The ERRORS section is rewritten, such that no error return values are now defined for this interface.
NAME
fflush — flush a stream

SYNOPSIS
#include <stdio.h>

int fflush(FILE *stream);

DESCRIPTION
If stream points to an output stream or an update stream in which the most recent operation was not input, fflush() causes any unwritten data for that stream to be written to the file, and the
st_ctime and st_mtime fields of the underlying file are marked for update.
If stream is a null pointer, fflush() performs this flushing action on all streams for which the
behaviour is defined above.

RETURN VALUE
Upon successful completion, fflush() returns 0. Otherwise, it returns EOF and sets errno to
indicate the error.

ERRORS
The fflush() function will fail if:

[EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying stream and the
process would be delayed in the write operation.
[EBADF] The file descriptor underlying stream is not valid.
[EFAULT] An attempt was made to write a file that exceeds the maximum file size or the
process’ file size limit.
[EINTR] The fflush() function was interrupted by a signal.
[EIO] The process is a member of a background process group attempting to write
to its controlling terminal, TOSTOP is set, the process is neither ignoring nor
blocking SIGTTOU and the process group of the process is orphaned. This
error may also be returned under implementation-dependent conditions.
[ENOMEM] There was no free space remaining on the device containing the file.
[EPipe] An attempt is made to write to a pipe or FIFO that is not open for reading by
any process. A SIGPIPE signal will also be sent to the process.

The fflush() function may fail if:

[ENXIO] A request was made of a non-existent device, or the request was outside the
device.

SEE ALSO
getrlimit(), ulimit(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
• The DESCRIPTION section is changed to describe the behaviour of fflush() if stream is a null
pointer.
The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- The following two paragraphs are withdrawn from the DESCRIPTION section (by POSIX as well as X/Open) because of the possibility of causing applications to malfunction, and the impossibility of implementing these mechanisms for pipes:

  If the stream is open for reading, any unread data buffered in the stream is discarded.

  For a stream open for reading, if the file is not already at EOF, and the file is one capable of seeking, the file offset of the underlying open file description is adjusted so that the next operation on the open file description deals with the byte after the last one read from, or written to, the stream being flushed.

- The [EBFBI] error is marked to indicate the extensions.
NAME
  ffs — find first set bit

SYNOPSIS
  #include <strings.h>

  int ffs(int i);

DESCRIPTION
  The ffs() function finds the first set bit (beginning with the least significant bit) and returns the
  index of that bit. Bits are numbered starting at one (the least significant bit).

RETURN VALUE
  The ffs() function returns the index of the first bit set. If i is 0, then ffs() returns 0.

ERRORS
  No errors are defined.

SEE ALSO
  <strings.h>.

CHANGE HISTORY
  First released in Issue 4, Version 2.
fgetc()  

NAME
fgetc — get a byte from a stream

SYNOPSIS
#include <stdio.h>

int fgetc(FILE *stream);

DESCRIPTION
The fgetc() function obtains the next byte (if present) as an unsigned char converted to an int, from the input stream pointed to by stream, and advances the associated file position indicator for the stream (if defined).

The fgetc() function may mark the st_atime field of the file associated with stream for update. The st_atime field will be marked for update by the first successful execution of fgetc(), fgets(), fgetwc(), fgetws(), fread(), fscanf(), getc(), getchar(), gets() or scanf() using stream that returns data not supplied by a prior call to ungetc() or ungetwc().

RETURN VALUE
Upon successful completion, fgetc() returns the next byte from the input stream pointed to by stream. If the stream is at end-of-file, the end-of-file indicator for the stream is set and fgetc() returns EOF. If a read error occurs, the error indicator for the stream is set, fgetc() returns EOF and sets errno to indicate the error.

ERRORS
The fgetc() function will fail if data needs to be read and:

[EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying stream and the process would be delayed in the fgetc() operation.

[EBADF] The file descriptor underlying stream is not a valid file descriptor open for reading.

[EINTR] The read operation was terminated due to the receipt of a signal, and no data was transferred.

UX [EIO] A physical I/O error has occurred, or the process is in a background process group attempting to read from its controlling terminal, and either the process is ignoring or blocking the SIGTTIN signal or the process group is orphaned. This error may also be generated for implementation-dependent reasons.

The fgetc() function may fail if:

EX [ENOMEM] Insufficient storage space is available.

EX [ENXIO] A request was made of a non-existent device, or the request was outside the capabilities of the device.

APPLICATION USAGE
If the integer value returned by fgetc() is stored into a variable of type char and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a variable of type char on widening to integer is implementation-dependent.

The ferror() or feof() functions must be used to distinguish between an error condition and an end-of-file condition.

SEE ALSO
feof(), ferror(), fopen(), getwchar(), getc(), <stdio.h>.
CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

- In the DESCRIPTION section:
  — The text is changed to make it clear that the function returns a byte value.
  — The list of functions that may cause the st_atime field to be updated is revised.
- In the ERRORS section, text is added to indicate that error returns will only be generated when data needs to be read into the stream buffer.
- Also in the ERRORS section, in previous issues generation of the [EIO] error depended on whether or not an implementation supported Job Control. This functionality is now defined as mandatory.
- The [ENXIO] and [ENOMEM] errors are marked as extensions.
- In the APPLICATION USAGE section, text is added to indicate how an application can distinguish between an error condition and an end-of-file condition.
- The description of [EINTR] is amended.

Issue 4, Version 2
In the ERRORS section, the description of [EIO] is updated to include the case where a physical I/O error occurs.
fgetpos() 

NAME
fgetpos — get current file position information

SYNOPSIS
#include <stdio.h>
int fgetpos(FILE *stream, fpos_t *pos);

DESCRIPTION
The fgetpos() function stores the current value of the file position indicator for the stream pointed to by stream in the object pointed to by pos. The value stored contains unspecified information usable by fsetpos() for repositioning the stream to its position at the time of the call to fgetpos().

RETURN VALUE
Upon successful completion, fgetpos() returns 0. Otherwise, it returns a non-zero value and sets errno to indicate the error.

ERRORS
The fgetpos() function may fail if:

EX
[EBADF] The file descriptor underlying stream is not valid.
[ESPIPE] The file descriptor underlying stream is associated with a pipe or FIFO.

SEE ALSO
fopen(), ftell(), rewind(), ungetc(), <stdio.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the ISO C standard.
NAME
fgets — get a string from a stream

SYNOPSIS
#include <stdio.h>
char *fgets(char *s, int n, FILE *stream);

DESCRIPTION
The `fgets()` function reads bytes from `stream` into the array pointed to by `s`, until `n−1` bytes are read, or a newline character is read and transferred to `s`, or an end-of-file condition is encountered. The string is then terminated with a null byte.

The `fgets()` function may mark the `st_atime` field of the file associated with `stream` for update. The `st_atime` field will be marked for update by the first successful execution of `fgetc()`, `fgets()`, `fgetwc()`, `fgetws()`, `fread()`, `fscanf()`, `getc()`, `getchar()`, `gets()` or `scanf()` using `stream` that returns data not supplied by a prior call to `ungetc()` or `ungetwc()`.

RETURN VALUE
Upon successful completion, `fgets()` returns `s`. If the stream is at end-of-file, the end-of-file indicator for the stream is set and `fgets()` returns a null pointer. If a read error occurs, the error indicator for the stream is set, `fgets()` returns a null pointer and sets `errno` to indicate the error.

ERRORS
Refer to `fgetc()`.

SEE ALSO
`fopen()`, `fread()`, `gets()`, `<stdio.h>`.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:

- In the DESCRIPTION section, (a) the text is changed to make it clear that the function reads bytes rather than (possibly multi-byte) characters, and (b) the list of functions that may cause the `st_atime` field to be updated is revised.
NAME
fgetwc — get a wide-character code from a stream

SYNOPSIS

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

wint_t fgetwc(FILE *stream);
```

DESCRIPTION
The `fgetwc()` function obtains the next character (if present) from the input stream pointed to by `stream`, converts that to the corresponding wide-character code and advances the associated file position indicator for the stream (if defined).

If an error occurs, the resulting value of the file position indicator for the stream is indeterminate.

The `fgetwc()` function may mark the `st_atime` field of the file associated with `stream` for update. The `st_atime` field will be marked for update by the first successful execution of `fgetc()`, `fgets()`, `fgetwc()`, `fgetws()`, `fread()`, `fscanf()`, `getc()`, `getchar()`, `gets()` or `scanf()` using `stream` that returns data not supplied by a prior call to `ungetc()` or `ungetwc()`.

RETURN VALUE
Upon successful completion the `fgetwc()` function returns the wide-character code of the character read from the input stream pointed to by `stream` converted to a type `wint_t`. If the stream is at end-of-file, the end-of-file indicator for the stream is set and `fgetwc()` returns WEOF.

If a read error occurs, the error indicator for the stream is set, `fgetwc()` returns WEOF and sets `errno` to indicate the error.

ERRORS
The `fgetwc()` function will fail if data needs to be read and:

- **[EAGAIN]** The O_NONBLOCK flag is set for the file descriptor underlying `stream` and the process would be delayed in the `fgetwc()` operation.
- **[EBADF]** The file descriptor underlying `stream` is not a valid file descriptor open for reading.
- **[EINTR]** The read operation was terminated due to the receipt of a signal, and no data was transferred.
- **[EIO]** A physical I/O error has occurred, or the process is in a background process group attempting to read from its controlling terminal, and either the process is ignoring or blocking the SIGTTIN signal or the process group is orphaned. This error may also be generated for implementation-dependent reasons.

UX

The `fgetwc()` function may fail if:

- **[ENOMEM]** Insufficient storage space is available.
- **[ENXIO]** A request was made of a non-existent device, or the request was outside the capabilities of the device.
- **[EILSEQ]** The data obtained from the input stream does not form a valid character.

APPLICATION USAGE
The `ferror()` or `feof()` functions must be used to distinguish between an error condition and an end-of-file condition.
SEE ALSO
  
  `feof()`, `ferror()`, `fopen()`, `<stdio.h>`, `<wchar.h>`.

CHANGE HISTORY
  
  First released in Issue 4.

  Derived from the MSE working draft.

Issue 4, Version 2
  
  In the `ERRORS` section, the description of [EIO] is updated to include the case where a physical I/O error occurs.
NAME
fgetws — get a wide character string from a stream

SYNOPSIS
OH
#include <stdio.h>
WP
#include <wchar.h>

wchar_t *fgetws(wchar_t *ws, int n, FILE *stream);

DESCRIPTION
The fgetws() function reads characters from the stream, converts these to the corresponding wide-character codes, places them in the wchar_t array pointed to by ws, until n−1 characters are read, or a newline character is read, converted and transferred to ws, or an end-of-file condition is encountered. The wide character string, ws, is then terminated with a null wide-character code.

If an error occurs, the resulting value of the file position indicator for the stream is indeterminate.

The fgetws() function may mark the st_atime field of the file associated with stream for update. The st_atime field will be marked for update by the first successful execution of fgetc(), fgets(), fgetwc(), fgetws(), fread(), fscanf(), getc(), getchar(), gets() or scanf() using stream that returns data not supplied by a prior call to ungetc() or ungetwc().

RETURN VALUE
Upon successful completion, fgetws() returns ws. If the stream is at end-of-file, the end-of-file indicator for the stream is set and fgetws() returns a null pointer. If a read error occurs, the error indicator for the stream is set, fgetws() returns a null pointer and sets errno to indicate the error.

ERRORS
Refer to fgetwc().

SEE ALSO
fopen(), fread(), <stdio.h>, <wchar.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
fileno — map stream pointer to file descriptor

SYNOPSIS
#include <stdio.h>

int fileno(FILE *stream);

DESCRIPTION
The fileno() function returns the integer file descriptor associated with the stream pointed to by stream.

RETURN VALUE
Upon successful completion, fileno() returns the integer value of the file descriptor associated with stream. Otherwise, the value −1 is returned and errno is set to indicate the error.

ERRORS
The fileno() function may fail if:

EX
[EBADF] The stream argument is not a valid stream.

SEE ALSO
fdopen(), fopen(), stdin, <stdio.h>, Section 2.4.1 on page 32.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:
• The [EBADF] error is marked as an extension.
NAME
color( ) — color function

SYNOPSIS
#include <string.h>

std::color(color_t color);

DESCRIPTION
The color() function returns the last character in a string.

ERRORS
The color() function will fail if:

[EINVAL] The result would cause an invalid color.

APPLICATION USAGE
The color() function may be used to change the color of text.

SEE ALSO
color(), isdigit(), std::string.
• The word `long` has been replaced with the words `long int` in the APPLICATION USAGE section.
• The return value specified for [EDOM] is marked as an extension.
fmod()  BASE  System Interfaces

NAME
fmod — floating-point remainder value function

SYNOPSIS
#include <math.h>

double fmod(double x, double y);

DESCRIPTION
The fmod() function returns the floating-point remainder of the division of x by y.

RETURN VALUE
The fmod() function returns the value x — i * y, for some integer i such that, if y is non-zero, the result has the same sign as x and magnitude less than the magnitude of y.

EX  If x or y is NaN, NaN is returned and errno may be set to [EDOM].
EX  If y is 0, NaN is returned and errno is set to [EDOM], or 0 is returned and errno may be set to [EDOM].
EX  If x is ±Inf, either 0 is returned and errno is set to [EDOM], or NaN is returned and errno may be set to [EDOM].

If y is non-zero, fmod(±0,y) returns the value of x. If x is not ±Inf, fmod(x,±Inf) returns the value of x.

If the result underflows, 0 is returned and errno may be set to [ERANGE].

ERRORS
The fmod() function may fail if:

EX  [EDOM] One or both of the arguments is NaN, or y is 0, or x is ±Inf.
EX  [ERANGE] The result underflows.
EX  No other errors will occur.

APPLICATION USAGE
Portable applications should not call fmod() with y equal to 0, because the result is implementation-dependent. The application should verify y is non-zero before calling fmod().

An application wishing to check for error situations should set errno to 0 before calling fmod(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

SEE ALSO
isnan(), <math.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:
- References to matherr() are removed.
- The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the ISO C standard and to rationalise error handling in the mathematics functions.
- The return value specified for [EDOM] is marked as an extension.
NAME
fmtmsg — display a message in the specified format on standard error and/or a system console

SYNOPSIS
#include <fmtmsg.h>

int fmtmsg(long classification, const char *label, int severity,
const char *text, const char *action, const char *tag);

DESCRIPTION
The fmtmsg() function can be used to display messages in a specified format instead of the traditional printf() function.

Based on a message’s classification component, fmtmsg() writes a formatted message either to standard error, to the console, or to both.

A formatted message consists of up to five components as defined below. The component classification is not part of a message displayed to the user, but defines the source of the message and directs the display of the formatted message.

classification
Contains identifiers from the following groups of major classifications and subclassifications. Any one identifier from a subclass may be used in combination with a single identifier from a different subclass. Two or more identifiers from the same subclass should not be used together, with the exception of identifiers from the display subclass. (Both display subclass identifiers may be used so that messages can be displayed to both standard error and the system console).

Major Classifications
Identifies the source of the condition. Identifiers are: MM_HARD (hardware), MM_SOFT (software), and MM_FIRM (firmware).

Message Source Subclassifications
Identifies the type of software in which the problem is detected. Identifiers are: MM_APPL (application), MM_UTIL (utility), and MM_OPSYS (operating system).

Display Subclassifications
Indicates where the message is to be displayed. Identifiers are: MM_PRINT to display the message on the standard error stream, MM_CONSOLE to display the message on the system console. One or both identifiers may be used.

Status Subclassifications
Indicates whether the application will recover from the condition. Identifiers are: MM_RECOVER (recoverable) and MM_NRECOV (non-recoverable).

An additional identifier, MM_NULLMC, indicates that no classification component is supplied for the message.

label
Identifies the source of the message. The format is two fields separated by a colon. The first field is up to 10 bytes, the second is up to 14 bytes.

severity
Indicates the seriousness of the condition. Identifiers for the levels of severity are:

MM_HALT Indicates that the application has encountered a severe fault and is halting. Produces the string "HALT".
MM_ERROR Indicates that the application has detected a fault. Produces the string "ERROR".

MM_WARNING Indicates a condition that is out of the ordinary, that might be a problem, and should be watched. Produces the string "WARNING".

MM_INFO Provides information about a condition that is not in error. Produces the string "INFO".

MM_NOSEV Indicates that no severity level is supplied for the message.

text Describes the error condition that produced the message. The character string is not limited to a specific size. If the character string is empty, then the text produced is unspecified.

action Describes the first step to be taken in the error-recovery process. The fmtmsg() function precedes the action string with the prefix: "TO FIX:". The action string is not limited to a specific size.

tag An identifier that references on-line documentation for the message. Suggested usage is that tag includes the label and a unique identifying number. A sample tag is "XSI:cat:146".

The MSGVERB environment variable (for message verbosity) tells fmtmsg() which message components it is to select when writing messages to standard error. The value of MSGVERB is a colon-separated list of optional keywords. Valid keywords are: label, severity, text, action, and tag. If MSGVERB contains a keyword for a component and the component’s value is not the component’s null value, fmtmsg() includes that component in the message when writing the message to standard error. If MSGVERB does not include a keyword for a message component, that component is not included in the display of the message. The keywords may appear in any order. If MSGVERB is not defined, if its value is the null string, if its value is not of the correct format, or if it contains keywords other than the valid ones listed above, fmtmsg() selects all components.

MSGVERB affects only which components are selected for display to standard error. All message components are included in console messages.

RETURN VALUE
The fmtmsg() function returns one of the following values:

- MM_OK The function succeeded.
- MM_NOTOK The function failed completely.
- MM_NOMSG The function was unable to generate a message on standard error, but otherwise succeeded.
- MM_NOCON The function was unable to generate a console message, but otherwise succeeded.

APPLICATION USAGE
One or more message components may be systematically omitted from messages generated by an application by using the null value of the argument for that component.

EXAMPLE
Example 1:
The following example of fmtmsg():
fmtmsg(MM_PRINT, "XSI:cat", MM_ERROR, "illegal option", "refer to cat in user’s reference manual", "XSI:cat:001")

produces a complete message in the specified message format:

XSI:cat: ERROR: illegal option
TO FIX: refer to cat in user’s reference manual XSI:cat:001

Example 2:
When the environment variable MSGVERB is set as follows:

MSGVERB=severity:text:action

and the Example 1 is used, fmtmsg() produces:

ERROR: illegal option
TO FIX: refer to cat in user’s reference manual

SEE ALSO
printf(), <fmtmsg.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
fnmatch — match filename or pathname

SYNOPSIS
#include <fnmatch.h>

int fnmatch(const char *pattern, const char *string, int flags);

DESCRIPTION
The fnmatch() function matches patterns as described in the XCU specification, Section 2.13.1, Patterns Matching a Single Character, and Section 2.13.2, Patterns Matching Multiple Characters. It checks the string specified by the string argument to see if it matches the pattern specified by the pattern argument.

The flags argument modifies the interpretation of pattern and string. It is the bitwise inclusive OR of zero or more of the flags defined in the header <fnmatch.h>. If the FNM_PATHNAME flag is set in flags, then a slash character in string will be explicitly matched by a slash in pattern; it will not be matched by either the asterisk or question-mark special characters, nor by a bracket expression. If the FNM_PATHNAME flag is not set, the slash character is treated as an ordinary character.

If FNM_NOESCAPE is not set in flags, a backslash character (\) in pattern followed by any other character will match that second character in string. In particular, \ will match a backslash in string. If FNM_NOESCAPE is set, a backslash character will be treated as an ordinary character.

If FNM_PERIOD is set in flags, then a leading period in string will match a period in pattern; as described by rule 2 in the XCU specification, Section 2.13.3, Patterns Used for Filename Expansion where the location of “leading” is indicated by the value of FNM_PATHNAME:

• If FNM_PATHNAME is set, a period is “leading” if it is the first character in string or if it immediately follows a slash.

• If FNM_PATHNAME is not set, a period is “leading” only if it is the first character of string.

If FNM_PERIOD is not set, then no special restrictions are placed on matching a period.

RETURN VALUE
If string matches the pattern specified by pattern, then fnmatch() returns 0. If there is no match, fnmatch() returns FNM_NOMATCH, which is defined in the header <fnmatch.h>. If an error occurs, fnmatch() returns another non-zero value.

ERRORS
No errors are defined.

APPLICATION USAGE
The fnmatch() function has two major uses. It could be used by an application or utility that needs to read a directory and apply a pattern against each entry. The find utility is an example of this. It can also be used by the pax utility to process its pattern operands, or by applications that need to match strings in a similar manner.

The name fnmatch() is intended to imply filename match, rather than pathname match. The default action of this function is to match filenames, rather than pathnames, since it gives no special significance to the slash character. With the FNM_PATHNAME flag, fnmatch() does match pathnames, but without tilde expansion, parameter expansion, or special treatment for period at the beginning of a filename.

SEE ALSO
glob(), wordexp(), <fnmatch.h>, the XCU specification.
CHANGE HISTORY

First released in Issue 4.

Derived from the ISO POSIX-2 standard.
NAME
fopen — open a stream

SYNOPSIS
#include <stdio.h>

FILE *fopen(const char * filename, const char * mode);

DESCRIPTION
The fopen() function opens the file whose pathname is the string pointed to by filename, and
associates a stream with it.

The argument mode points to a string beginning with one of the following sequences:

- **r** or **rb** | open file for reading
- **w** or **wb** | truncate to zero length or create file for writing
- **a** or **ab** | append; open or create file for writing at end-of-file
- **r+** or **rb+** or **r+b** | open file for update (reading and writing)
- **w+** or **wb+** or **w+b** | truncate to zero length or create file for update
- **a+** or **ab+** or **a+b** | append; open or create file for update, writing at end-of-file

The character b has no effect, but is allowed for ISO C standard conformance. Opening a file
with read mode (r as the first character in the mode argument) fails if the file does not exist or
cannot be read.

Opening a file with append mode (a as the first character in the mode argument) causes all
subsequent writes to the file to be forced to the then current end-of-file, regardless of intervening
calls to fseek().

When a file is opened with update mode (+ as the second or third character in the mode argument), both input and output may be performed on the associated stream. However,
output must not be directly followed by input without an intervening call to fflush() or to a file
positioning function (fseek(), fsetpos() or rewind()), and input must not be directly followed by
output without an intervening call to a file positioning function, unless the input operation
encounters end-of-file.

When opened, a stream is fully buffered if and only if it can be determined not to refer to an
interactive device. The error and end-of-file indicators for the stream are cleared.

If mode is **w**, **a**, **w+** or **a+** and the file did not previously exist, upon successful completion,
fopen() function will mark for update the st_atime, st_ctime and st_mtime fields of the file and the
st_ctime and st_mtime fields of the parent directory.

If mode is **w** or **w+** and the file did previously exist, upon successful completion, fopen() will
mark for update the st_ctime and st_mtime fields of the file. The fopen() function will allocate a
file descriptor as open() does.

RETURN VALUE
Upon successful completion, fopen() returns a pointer to the object controlling the stream.
Otherwise, a null pointer is returned, and errno is set to indicate the error.

ERRORS
The fopen() function will fail if:

- **[EACCES]** Search permission is denied on a component of the path prefix, or the file
  exists and the permissions specified by mode are denied, or the file does not
  exist and write permission is denied for the parent directory of the file to be
  created.
The fopen() function may fail if:

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

- **[EMFILE]** {FOPEN_MAX} streams are currently open in the calling process.

- **[EMFILE]** {STREAM_MAX} streams are currently open in the calling process.

- **[ENOMEM]** Insufficient storage space is available.

- **[ETXTBSY]** The file is a pure procedure (shared text) file that is being executed and `mode` requires write access.

**APPLICATION USAGE**

{STREAM_MAX} is the number of streams that one process can have open at one time. If defined, it has the same value as {FOPEN_MAX}.

**SEE ALSO**

`fclose()`, `fdopen()`, `freopen()`, `<stdio.h>`.

**CHANGE HISTORY**

First released in Issue 1.

Derived from Issue 1 of the SVID.

**Issue 4**

The following changes are incorporated for alignment with the ISO C standard:

- The type of arguments `filename` and `mode` are changed from `char *` to `const char *`.

- In the `DESCRIPTION` section, (a) the use and settings of the `mode` argument are changed to support binary streams and (b) `setpos()` is added to the list of file positioning functions.
The following change is incorporated for alignment with the FIPS requirements:

- In the **ERRORS** section, the condition whereby [ENAMETOOLONG] will be returned if a
  pathname component is larger that [NAME_MAX] is now defined as mandatory and marked
  as an extension.

Other changes are incorporated as follows:

- In the **DESCRIPTION** section the descriptions of input and output operations on update
  streams are changed to be requirements on the application.

- The [EMFILE] error is added to the **ERRORS** section, and all the optional errors are marked
  as extensions.

### Issue 4, Version 2

The **ERRORS** section is updated for X/OPEN UNIX conformance as follows:

- It states that [ELOOP] will be returned if too many symbolic links are encountered during
  pathname resolution.

- A second [ENAMETOOLONG] condition is defined that may report excessive length of an
  intermediate result of pathname resolution of a symbolic link.
NAME
fork — create a new process

SYNOPSIS
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);

DESCRIPTION
The fork() function creates a new process. The new process (child process) is an exact copy of the calling process (parent process) except as detailed below.

• The child process has a unique process ID.
• The child process ID also does not match any active process group ID.
• The child process has a different parent process ID (that is, the process ID of the parent process).
• The child process has its own copy of the parent’s file descriptors. Each of the child’s file descriptors refers to the same open file description with the corresponding file descriptor of the parent.
• The child process has its own copy of the parent’s open directory streams. Each open directory stream in the child process may share directory stream positioning with the corresponding directory stream of the parent.

EX
• The child process may have its own copy of the parent’s message catalogue descriptors.
• The child process’ values of tms_utime, tms_stime, tms_cutime and tms_cstime are set to 0.
• The time left until an alarm clock signal is reset to 0.

EX
• All semadj values are cleared.
• File locks set by the parent process are not inherited by the child process.
• The set of signals pending for the child process is initialised to the empty set.

UX
• Interval timers are reset in the child process.

The inheritance of process characteristics not defined by this document is implementation-dependent. After fork(), both the parent and the child processes are capable of executing independently before either one terminates.

RETURN VALUE
Upon successful completion, fork() returns 0 to the child process and returns the process ID of the child process to the parent process. Otherwise, −1 is returned to the parent process, no child process is created, and errno is set to indicate the error.

ERRORS
The fork() function will fail if:

[EAGAIN] The system lacked the necessary resources to create another process, or the system-imposed limit on the total number of processes under execution system-wide or by a single user [CHILD_MAX] would be exceeded.

The fork() function may fail if:

[ENOMEM] Insufficient storage space is available.
SEE ALSO

alarm(), exec, fcntl(), semop(), signal(), times(), <sys/types.h>, <unistd.h>.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated for alignment with the ISO POSIX-1 standard:

• The argument list is explicitly defined as void.

• Though functionally identical to Issue 3, the DESCRIPTION section has been reorganised to improve clarity and to align more closely with the ISO POSIX-1 standard.

• The description of the [EAGAIN] error is updated to indicate that this error can also be returned if a system lacks the resources to create another process.

Another change is incorporated as follows:

• The header <sys/types.h> is now marked as optional (OH); this header need not be included on XSI-conformant systems.

Issue 4, Version 2

The DESCRIPTION is changed for X/Open UNIX conformance to identify that interval timers are reset in the child process.
NAME
fpathconf — get configurable pathname variables

SYNOPSIS
#include <unistd.h>
long int fpathconf(int fildes, int name);

DESCRIPTION
Refer to pathconf().

CHANGE HISTORY
First released in Issue 3.
Entry included for alignment with the POSIX.1-1988 standard.

Issue 4
The function now has the full long int return type in the SYNOPSIS section.
NAME
fprintf, printf, sprintf — print formatted output

SYNOPSIS
#include <stdio.h>

int fprintf(FILE *stream, const char *format, ...);
int printf(const char *format, ...);
int sprintf(char *s, const char *format, ...);

DESCRIPTION
The fprintf() function places output on the named output stream. The printf() function places output on the standard output stream stdout. The sprintf() function places output followed by the null byte, \0, in consecutive bytes starting at *s; it is the user's responsibility to ensure that enough space is available.

Each of these functions converts, formats and prints its arguments under control of the format. The format is a character string, beginning and ending in its initial shift state, if any. The format is composed of zero or more directives: ordinary characters, which are simply copied to the output stream and conversion specifications, each of which results in the fetching of zero or more arguments. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

EX Conversions can be applied to the nth argument after the format in the argument list, rather than to the next unused argument. In this case, the conversion character % (see below) is replaced by the sequence %n$, where n is a decimal integer in the range [1, [NL_ARGMAX]], giving the position of the argument in the argument list. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages (see the EXAMPLES section).

In format strings containing the %n$ form of conversion specifications, numbered arguments in the argument list can be referenced from the format string as many times as required.

In format strings containing the % form of conversion specifications, each argument in the argument list is used exactly once.

All forms of the fprintf() functions allow for the insertion of a language-dependent radix character in the output string. The radix character is defined in the program's locale (category LC_NUMERIC). In the POSIX locale, or in a locale where the radix character is not defined, the radix character defaults to a period (.).

EX Each conversion specification is introduced by the % character or by the character sequence %n$, after which the following appear in sequence:

• Zero or more flags (in any order), which modify the meaning of the conversion specification.

• An optional minimum field width. If the converted value has fewer bytes than the field width, it will be padded with spaces by default on the left; it will be padded on the right, if the left-adjustment flag (-), described below, is given to the field width. The field width takes the form of an asterisk (*), described below, or a decimal integer.

• An optional precision that gives the minimum number of digits to appear for the d, i, o, u, x and X conversions; the number of digits to appear after the radix character for the e, E and f conversions; the maximum number of significant digits for the g and G conversions; or the maximum number of bytes to be printed from a string in s and S conversions. The precision takes the form of a period (.) followed either by an asterisk (*), described below, or an
optional decimal digit string, where a null digit string is treated as 0. If a precision appears with any other conversion character, the behaviour is undefined.

- An optional h specifying that a following d, i, o, u, x or X conversion character applies to a type short int or type unsigned short int argument (the argument will have been promoted according to the integral promotions, and its value will be converted to type short int or unsigned short int before printing); an optional h specifying that a following n conversion character applies to a pointer to a type short int argument; an optional l (ell) specifying that a following d, i, o, u, x or X conversion character applies to a type long int or unsigned long int argument; an optional l (ell) specifying that a following n conversion character applies to a pointer to a type long int argument; or an optional L specifying that a following e, E, f, g or G conversion character applies to a type long double argument. If an h, l or L appears with any other conversion character, the behaviour is undefined.

- A conversion character that indicates the type of conversion to be applied.

A field width, or precision, or both, may be indicated by an asterisk (*). In this case an argument of type int supplies the field width or precision. Arguments specifying field width, or precision, or both must appear in that order before the argument, if any, to be converted. A negative field width is taken as a − flag followed by a positive field width. A negative precision is taken as if the precision were omitted. In format strings containing the %n$ form of a conversion specification, a field width or precision may be indicated by the sequence *m$, where m is a decimal integer in the range \([1, \text{NL_ARGMAX}]\) giving the position in the argument list (after the format argument) of an integer argument containing the field width or precision, for example:

```
printf("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```

The format can contain either numbered argument specifications (that is, %n$ and *m$), or unnumbered argument specifications (that is, % and *), but normally not both. The only exception to this is that % can be mixed with the %n$ form. The results of mixing numbered and unnumbered argument specifications in a format string are undefined. When numbered argument specifications are used, specifying the Nth argument requires that all the leading arguments, from the first to the \((N-1)\)th, are specified in the format string.

The flag characters and their meanings are:

- The result of the conversion will be left-justified within the field. The conversion will be right-justified if this flag is not specified.

+ The result of a signed conversion will always begin with a sign (+ or −). The conversion will begin with a sign only when a negative value is converted if this flag is not specified.

space If the first character of a signed conversion is not a sign or if a signed conversion results in no characters, a space will be prefixed to the result. This means that if the space and + flags both appear, the space flag will be ignored.

# This flag specifies that the value is to be converted to an alternative form. For o conversion, it increases the precision (if necessary) to force the first digit of the result to be 0. For x or X conversions, a non-zero result will have 0x (or 0X) prefixed to it. For e, E, f, g or G conversions, the result will always contain a radix character, even if no digits follow the radix character. Without this flag, a radix character appears in the result of these conversions only if a digit follows it. For g and G conversions, trailing
zeros will not be removed from the result as they normally are. For other conversions, the behaviour is undefined.

0 For d, i, o, u, x, X, e, E, f, g and G conversions, leading zeros (following any indication of sign or base) are used to pad to the field width; no space padding is performed. If the 0 and – flags both appear, the 0 flag will be ignored. For d, i, o, u, x and X conversions, if a precision is specified, the 0 flag will be ignored. If the 0 and ’ flags both appear, the grouping characters are inserted before zero padding. For other conversions, the behaviour is undefined.

The conversion characters and their meanings are:

d, i The int argument is converted to a signed decimal in the style [−]ddd. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting 0 with an explicit precision of 0 is no characters.

0 The unsigned int argument is converted to unsigned octal format in the style dddd. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting 0 with an explicit precision of 0 is no characters.

u The unsigned int argument is converted to unsigned decimal format in the style dddd. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting 0 with an explicit precision of 0 is no characters.

x The unsigned int argument is converted to unsigned hexadecimal format in the style dddd; the letters abcdef are used. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting 0 with an explicit precision of 0 is no characters.

X Behaves the same as the x conversion character except that letters ABCDEF are used instead of abcdef.

f The double argument is converted to decimal notation in the style [−]ddd.ddd, where the number of digits after the radix character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is explicitly 0 and no # flag is present, no radix character appears. If a radix character appears, at least one digit appears before it. The value is rounded to the appropriate number of digits.

EX The fprintf() family of functions may make available character string representations for infinity and NaN.

e, E The double argument is converted in the style [−]d.ddde±dd, where there is one digit before the radix character (which is non-zero if the argument is non-zero) and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is 0 and no # flag is present, no radix character appears. The value is rounded to the appropriate number of digits. The E conversion character will produce a number with E instead of e introducing the exponent. The exponent always contains at least two digits. If the value is 0, the exponent is 0.

EX The fprintf() family of functions may make available character string representations for infinity and NaN.
The double argument is converted in the style f or e (or in the style E in the case of a G conversion character), with the precision specifying the number of significant digits. If an explicit precision is 0, it is taken as 1. The style used depends on the value converted; style e (or E) will be used only if the exponent resulting from such a conversion is less than \(-4\) or greater than or equal to the precision. Trailing zeros are removed from the fractional portion of the result; a radix character appears only if it is followed by a digit.

EX

The \texttt{fprintf()} family of functions may make available character string representations for infinity and NaN.

EX

The \texttt{int} argument is converted to an unsigned char, and the resulting byte is written.

EX

The argument must be a pointer to an array of char. Bytes from the array are written up to (but not including) any terminating null byte. If the precision is specified, no more than that many bytes are written. If the precision is not specified or is greater than the size of the array, the array must contain a null byte.

EX

The argument must be a pointer to \texttt{void}. The value of the pointer is converted to a sequence of printable characters, in an implementation-dependent manner.

EX

The argument must be a pointer to an integer into which is written the number of bytes written to the output so far by this call to one of the \texttt{fprintf()} functions. No argument is converted.

EX

The wchar_t argument is converted to an array of bytes representing a character, and the resulting character is written. The conversion is the same as that expected from \texttt{wctomb()}. In a locale with state-dependent encoding the behaviour with regard to the stream’s shift state is implementation-dependent.

EX

The argument must be a pointer an array of type wchar_t. Wide character codes from the array, up to but not including any terminating null wide-character code are converted to a sequence of bytes, and the resulting bytes are written. If the precision is specified no more than that many bytes are written and only complete characters are written. If the precision is not specified, or is greater than the size of the array of converted bytes, the array of wide characters must be terminated by a null wide character. The conversion is the same as that expected from \texttt{wcstombs()}. In a locale with state-dependent encoding the behaviour with regard to the stream’s shift state is implementation-dependent.

% Print a %; no argument is converted. The entire conversion specification must be %%.

If a conversion specification does not match one of the above forms, the behaviour is undefined.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by \texttt{fprintf()} and \texttt{printf()} are printed as if \texttt{fputc()} had been called.

The \texttt{st_mtime} and \texttt{st_ctime} fields of the file will be marked for update between the call to a successful execution of \texttt{fprintf()} or \texttt{printf()} and the next successful completion of a call to \texttt{fflush()} or \texttt{fclose()} on the same stream or a call to \texttt{exit()} or \texttt{abort()}.

**RETURN VALUE**

Upon successful completion, these functions return the number of bytes transmitted excluding the terminating null in the case of \texttt{sprintf()} or a negative value if an output error was encountered.
fprintf()  

ERRORS

For the conditions under which fprintf() and printf() will fail and may fail, refer to fputc() or fputwc().

In addition, all forms of fprintf() may fail:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[EILSEQ]</td>
<td>A wide-character code that does not correspond to a valid character has been detected.</td>
</tr>
<tr>
<td>[EINVAL]</td>
<td>There are insufficient arguments.</td>
</tr>
</tbody>
</table>

In addition, printf() and fprintf() may fail if:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ENOMEM]</td>
<td>Insufficient storage space is available.</td>
</tr>
</tbody>
</table>

EXAMPLES

To print the language-independent date and time format, the following statement could be used:

```c
printf (format, weekday, month, day, hour, min);
```

For American usage, `format` could be a pointer to the string:

```
"%s, %s %d, %d:%.2d
"
```

producing the message:

```
Sunday, July 3, 10:02
```

whereas for German usage, `format` could be a pointer to the string:

```
"%1$s, %3$d. %2$s, %4$d:%5$.2d
"
```

producing the message:

```
Sonntag, 3. Juli, 10:02
```

APPLICATION USAGE

If the application calling fprintf() has any objects of type wchar_t, it must also include either `<sys/types.h>` or `<stddef.h>` to have wchar_t defined.

SEE ALSO

fputc(), fscanf(), setlocale(), `<stdio.h>`, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated for alignment with the ISO C standard:

- The type of the format arguments is changed from char * to const char *.
- The DESCRIPTION section is reworded or presented differently in a number of places for alignment with the ISO C standard, and also for clarity. There are no functional changes, except as noted elsewhere in this CHANGE HISTORY section.

The following changes are incorporated for alignment with the MSE working draft:

- The C and S conversion characters are added, indicating respectively a wide-character of type wchar_t and pointer to a wide-character string of type wchar_t* in the argument list.
Other changes are incorporated as follows:

- In the DESCRIPTION section, references to `langinfo` data are marked as extensions. The reference to `langinfo` data is removed from the description of the radix character.

- The ’ (single-quote) flag is added to the list of flag characters and marked as an extension. This flag directs that numeric conversion will be formatted with the decimal grouping character.

- The detailed description of this function is provided here instead of under `printf()`.

- The information in the APPLICATION USAGE section is moved to the DESCRIPTION section. A new APPLICATION USAGE section is added.

- The [EILSEQ] error is added to the ERRORS section and all errors are marked as extensions.

**Issue 4, Version 2**

The [ENOMEM] error is added to the ERRORS section as an optional error.
NAME
    fputc — put byte on a stream

SYNOPSIS
    #include <stdio.h>
    int fputc(int c, FILE *stream);

DESCRIPTION
    The fputc() function writes the byte specified by c (converted to an unsigned char) to the output stream pointed to by stream, at the position indicated by the associated file-position indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot support positioning requests, or if the stream was opened with append mode, the byte is appended to the output stream.

    The st_ctime and st_mtime fields of the file will be marked for update between the successful execution of fputc() and the next successful completion of a call to fflush() or fclose() on the same stream or a call to exit() or abort().

RETURN VALUE
    Upon successful completion, fputc() returns the value it has written. Otherwise, it returns EOF, the error indicator for the stream is set, and errno is set to indicate the error.

ERRORS
    The fputc() function will fail if either the stream is unbuffered or the stream’s buffer needs to be flushed, and:

    [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying stream and the process would be delayed in the write operation.

    [EBADF] The file descriptor underlying stream is not a valid file descriptor open for writing.

    [EFAULT] An attempt was made to write to a file that exceeds the maximum file size or the process’ file size limit.

    [EINTR] The write operation was terminated due to the receipt of a signal, and no data was transferred.

    [EIO] A physical I/O error has occurred, or the process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU and the process group of the process is orphaned. This error may also be returned under implementation-dependent conditions.

    [ENOSPC] There was no free space remaining on the device containing the file.

    [EPPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal will also be sent to the process.

    The fputc() function may fail if:

    [ENOMEM] Insufficient storage space is available.

    [ENXIO] A request was made of a non-existent device, or the request was outside the capabilities of the device.

SEE ALSO
    ferror(), fopen(), getrlimit(), putc(), puts(), setbuf(), ulimit(), <stdio.h>.
CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

- In the DESCRIPTION section, the text is changed to make it clear that the function writes byte values, rather than (possibly multi-byte) character values.
- In the ERRORS section, text is added to indicate that error returns will only be generated when either the stream is unbuffered, or if the stream buffer needs to be flushed.
- Also in the ERRORS section, in previous issues generation of the [EIO] error depended on whether or not an implementation supported Job Control. This functionality is now defined as mandatory.
- The [ENXIO] error is moved to the list of optional errors, and all the optional errors are marked as extensions.
- The description of [EINTR] is amended.
- The [EFBIG] error is marked to show extensions.

Issue 4, Version 2
In the ERRORS section, the description of [EIO] is updated to include the case where a physical I/O error occurs.
NAME
fputs — put a string on a stream

SYNOPSIS
#include <stdio.h>

int fputs(const char *s, FILE *stream);

DESCRIPTION
The fputs() function writes the null-terminated string pointed to by s to the stream pointed to by stream. The terminating null byte is not written.

The st_ctime and st_mtime fields of the file will be marked for update between the successful execution of fputs() and the next successful completion of a call to fflush() or fclose() on the same stream or a call to exit() or abort().

RETURN VALUE
Upon successful completion, fputs() returns a non-negative number. Otherwise it returns EOF, sets an error indicator for the stream and errno is set to indicate the error.

ERRORS
Refer to fputc().

APPLICATION USAGE
The puts() function appends a newline character while fputs() does not.

SEE ALSO
fopen(), putc(), puts(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:

- The type of argument s is changed from char * to const char *.

Another change is incorporated as follows:

- In the DESCRIPTION section, the words “null character” are replaced by “null byte”, to make it clear that this interface deals solely in byte values.
NAME
fputwc — put wide-character code on a stream

SYNOPSIS

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

wint_t fputwc(wint_t wc, FILE *stream);
```

DESCRIPTION
The `fputwc()` function writes the character corresponding to the wide-character code `wc` to the output stream pointed to by `stream`, at the position indicated by the associated file-position indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot support positioning requests, or if the stream was opened with append mode, the character is appended to the output stream. If an error occurs whilst writing the character, the shift state of the output file is left in an undefined state.

The `st_ctime` and `st_mtime` fields of the file will be marked for update between the successful execution of `fputwc()` and the next successful completion of a call to `fflush()` or `fclose()` on the same stream or a call to `exit()` or `abort()`.

RETURN VALUE
Upon successful completion, `fputwc()` returns `wc`. Otherwise, it returns WEOF, the error indicator for the stream is set, and `errno` is set to indicate the error.

ERRORS
The `fputwc()` function will fail if either the stream is unbuffered or data in the `stream`'s buffer needs to be written, and:

- **[EAGAIN]** The O_NONBLOCK flag is set for the file descriptor underlying `stream` and the process would be delayed in the write operation.
- **[EBADF]** The file descriptor underlying `stream` is not a valid file descriptor open for writing.
- **[EFBIG]** An attempt was made to write to a file that exceeds the maximum file size or the process' file size limit.
- **[EINVAL]** The write operation was terminated due to the receipt of a signal, and no data was transferred.
- **[EIO]** A physical I/O error has occurred, or the process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU and the process group of the process is orphaned. This error may also be returned under implementation-dependent conditions.
- **[ENOSPC]** There was no free space remaining on the device containing the file.
- **[EPIPE]** An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal will also be sent to the process.
The `fputwc()` function may fail if:

- **[ENOMEM]** Insufficient storage space is available.
- **[ENXIO]** A request was made of a non-existent device, or the request was outside the capabilities of the device.
- **[EILSEQ]** The wide-character code *wc* does not correspond to a valid character.

**SEE ALSO**

`ferror()`, `fopen()`, `setbuf()`, `ulimit()`, `<stdio.h>`, `<wchar.h>`.  

**CHANGE HISTORY**

First released in Issue 4.

Derived from the MSE working draft.

**Issue 4, Version 2**

In the **ERRORS** section, the description of **[EIO]** is updated to include the case where a physical I/O error occurs.
NAME
fputws — put wide character string on a stream

SYNOPSIS

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

int fputws(const wchar_t * ws , FILE * stream);
```

DESCRIPTION

The `fputws()` function writes a character string corresponding to the (null-terminated) wide character string pointed to by `ws` to the stream pointed to by `stream`. No character corresponding to the terminating null wide-character code is written.

The `st_ctime` and `st_mtime` fields of the file will be marked for update between the successful execution of `fputws()` and the next successful completion of a call to `fflush()` or `fclose()` on the same stream or a call to `exit()` or `abort()`.

RETURN VALUE

Upon successful completion, `fputws()` returns a non-negative number. Otherwise it returns −1, sets an error indicator for the stream and `errno` is set to indicate the error.

ERRORS

Refer to `fputwc()`.

APPLICATION USAGE

The `fputws()` function does not append a newline character.

SEE ALSO

`fopen()`, `<stdio.h>`, `<wchar.h>`.

CHANGE HISTORY

First released in Issue 4.

Derived from the MSE working draft.
NAME
fread — binary input

SYNOPSIS
#include <stdio.h>

size_t fread(void *ptr, size_t size, size_t nitems, FILE *stream);

DESCRIPTION
The fread( ) function reads, into the array pointed to by ptr, up to nitems members whose size is specified by size in bytes, from the stream pointed to by stream. The file position indicator for the stream (if defined) is advanced by the number of bytes successfully read. If an error occurs, the resulting value of the file position indicator for the stream is indeterminate. If a partial member is read, its value is indeterminate.

The fread( ) function may mark the st_atime field of the file associated with stream for update. The st_atime field will be marked for update by the first successful execution of fgetc(), fgets(), fgetwc(), fgetws(), fread(), fscanf(), getc(), getchar(), gets() or scanf() using stream that returns data not supplied by a prior call to ungetc() or ungetwc().

RETURN VALUE
Upon successful completion, fread() returns the number of members successfully read which is less than nitems only if a read error or end-of-file is encountered. If size or nitems is 0, fread() returns 0 and the contents of the array and the state of the stream remain unchanged. Otherwise, if a read error occurs, the error indicator for the stream is set and errno is set to indicate the error.

ERRORS
Refer to fgetc().

APPLICATION USAGE
The ferror() or feof() functions must be used to distinguish between an error condition and an end-of-file condition.

SEE ALSO
feof(), ferror(), fopen(), getc(), getchar(), gets(), scanf(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:

• In the RETURN VALUE section, the behaviour if size or nitems is 0 is defined.

Another change is incorporated as follows:

• The list of functions that may cause the st_atime field to be updated is revised.
NAME
free — free allocated memory

SYNOPSIS
#include <stdlib.h>
void free(void * ptr);

DESCRIPTION
The free() function causes the space pointed to by ptr to be deallocated; that is, made available
for further allocation. If ptr is a null pointer, no action occurs. Otherwise, if the argument does
not match a pointer earlier returned by the malloc(), calloc(), realloc() or valloc() function, or if the
space is deallocated by a call to free() or realloc(), the behaviour is undefined.

Any use of a pointer that refers to freed space causes undefined behaviour.

RETURN VALUE
The free() function returns no value.

ERRORS
No errors are defined.

APPLICATION USAGE
There is now no requirement for the implementation to support the inclusion of <malloc.h>.

SEE ALSO
calloc(), malloc(), realloc(), valloc(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:

• The DESCRIPTION section now states that the behaviour is undefined if any use is made of
a pointer that refers to freed space. This was implied but not stated explicitly in Issue 3.

Another change is incorporated as follows:

• The APPLICATION USAGE section is changed to record that <malloc.h> need no longer be
supported on XSI-conformant systems.

Issue 4, Version 2
The DESCRIPTION is updated for X/OPEN UNIX conformance to indicate that the free() function can also be used to free memory allocated by valloc().
NAME
freopen — open a stream

SYNOPSIS
#include <stdio.h>
FILE *freopen(const char * filename , const char * mode , FILE * stream);

DESCRIPTION
The freopen() function first attempts to flush the stream and close any file descriptor associated
with stream. Failure to flush or close the file successfully is ignored. The error and end-of-file
indicators for the stream are cleared.

The freopen() function opens the file whose pathname is the string pointed to by filename and
associates the stream pointed to by stream with it. The mode argument is used just as in fopen().

The original stream is closed regardless of whether the subsequent open succeeds.

RETURN VALUE
Upon successful completion, freopen() returns the value of stream. Otherwise a null pointer is
returned and errno is set to indicate the error.

ERRORS
The freopen() function will fail if:

[EACCES] Search permission is denied on a component of the path prefix, or the file
exists and the permissions specified by mode are denied, or the file does not
exist and write permission is denied for the parent directory of the file to be
created.

[EINTR] A signal was caught during freopen().

[EISDIR] The named file is a directory and mode requires write access.

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

The freopen() function may fail if:

[ENAMETOOLONG] The length of the filename exceeds [PATH_MAX] or a pathname component is
longer than [NAME_MAX].

[ENFILE] The maximum allowable number of files is currently open in the system.

[ENOENT] A component of filename does not name an existing file or filename is an empty
string.

[ENOSPC] The directory or file system that would contain the new file cannot be
expanded, the file does not exist, and it was to be created.

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

[ENXIO] The named file is a character special or block special file, and the device
associated with this special file does not exist.

[EROFS] The named file resides on a read-only file system and mode requires write
access.

The freopen() function may fail if:

[ELOOP] Too many symbolic links were encountered in resolving path.

[EMFILE] [OPEN_MAX] file descriptors are currently open in the calling process.

[ENOENT] A component of filename does not name an existing file or filename is an empty
string.

[ENFILE] The maximum allowable number of files is currently open in the system.

[ENOSPC] The directory or file system that would contain the new file cannot be
expanded, the file does not exist, and it was to be created.

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

[ENXIO] The named file is a character special or block special file, and the device
associated with this special file does not exist.

[EROFS] The named file resides on a read-only file system and mode requires write
access.

The freopen() function may fail if:

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

**[ENAMETOOLONG]**
Pathname resolution of a symbolic link produced an intermediate result whose length exceeds \([\text{PATH_MAX}]\).

**[ENOMEM]**
Insufficient storage space is available.

**[ENXIO]**
A request was made of a non-existent device, or the request was outside the capabilities of the device.

**[ETXTBSY]**
The file is a pure procedure (shared text) file that is being executed and \(mode\) requires write access.

**APPLICATION USAGE**
The `freopen()` function is typically used to attach the preopened streams associated with `stdin`, `stdout` and `stderr` to other files.

**SEE ALSO**
`fclose()`, `fopen()`, `fdopen()`, `<stdio.h>`.

**CHANGE HISTORY**
First released in Issue 1.
Derived from Issue 1 of the SVID.

**Issue 4**
The following change is incorporated for alignment with the ISO C standard:
- The type of arguments `filename` and `mode` are changed from `char *` to `const char *`.

The following change is incorporated for alignment with the FIPS requirements:
- In the **ERRORS** section, the condition whereby \([\text{ENAMETOOLONG}]\) will be returned if a pathname component is larger that \([\text{Name_MAX}]\) is now defined as mandatory and marked as an extension.

Other changes are incorporated as follows:
- In the **DESCRIPTION**, the word “name” is replaced by “pathname”, to make it clear that the interface is not limited to accepting filenames only.
- In the **ERRORS** section, (a) the description of the \([\text{EINVAL}]\) error has been changed to refer to \([\text{OPEN_MAX}]\) file descriptors rather than \([\text{FOPEN_MAX}]\) file descriptors, directories and message catalogues, (b) the errors \([\text{EINVAL}]\), \([\text{ENOMEM}]\) and \([\text{ETXTBSY}]\) are marked as extensions, and (c) the \([\text{ENXIO}]\) error is added in the “may fail” section and marked as an extension.

**Issue 4, Version 2**
The **ERRORS** section is updated for X/Open UNIX conformance as follows:
- It states that \([\text{ELOOP}]\) will be returned if too many symbolic links are encountered during pathname resolution.
- A second \([\text{ENAMETOOLONG}]\) condition is defined that may report excessive length of an intermediate result of pathname resolution of a symbolic link.
NAME
frexp — extract mantissa and exponent from double precision number

SYNOPSIS
#include <math.h>

double frexp(double num, int *exp);

DESCRIPTION
The frexp() function breaks a floating-point number into a normalised fraction and an integral
power of 2. It stores the integer exponent in the int object pointed to by exp.

RETURN VALUE
The frexp() function returns the value x, such that x is a double with magnitude in the interval
\([\frac{1}{2}, 1)\) or 0, and num equals x times 2 raised to the power *exp.
If num is 0, both parts of the result are 0.

EX If num is NaN, NaN is returned, errno may be set to [EDOM] and the value of *exp is unspecified.
EX If num is \pm\Inf, num is returned, errno may be set to [EDOM] and the value of *exp is unspecified.

ERRORS
The frexp() function may fail if:

EX [EDOM] The value of num is NaN or \pm\Inf.
EX No other errors will occur.

APPLICATION USAGE
An application wishing to check for error situations should set errno to 0 before calling frexp(). If
errno is non-zero on return, or the return value is NaN, an error has occurred.

SEE ALSO
isnan(), ldexp(), modf(), <math.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:
• Removed references to matherr().
• The name of the first argument is changed from value to num.
• The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with
  the ISO C standard and to rationalise error handling in the mathematics functions.
• The return value specified for [EDOM] is marked as an extension.
NAME
fscanf, scanf, sscanf — convert formatted input

SYNOPSIS
#include <stdio.h>

int fscanf(FILE * stream , const char * format , ...);
int scanf(const char * format , ...);
int sscanf(const char * s , const char * format , ...);

DESCRIPTION
The fscanf() function reads from the named input stream. The scanf() function reads from the standard input stream stdin. The sscanf() function reads from the string s. Each function reads bytes, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string format described below, and a set of pointer arguments indicating where the converted input should be stored. The result is undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

Conversions can be applied to the nth argument after the format in the argument list, rather than to the next unused argument. In this case, the conversion character % (see below) is replaced by the sequence %n$, where n is a decimal integer in the range [1, [NL_ARGMAX]]. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages. In format strings containing the %n$ form of conversion specifications, it is unspecified whether numbered arguments in the argument list can be referenced from the format string more than once.

The format can contain either form of a conversion specification, that is, % or %n$, but the two forms cannot normally be mixed within a single format string. The only exception to this is that %% or %* can be mixed with the %n$ form.

The fscanf() function in all its forms allows for detection of a language-dependent radix character in the input string. The radix character is defined in the program’s locale (category LC_NUMERIC). In the POSIX locale, or in a locale where the radix character is not defined, the radix character defaults to a period (.).

The format is a character string, beginning and ending in its initial shift state, if any, composed of zero or more directives. Each directive is composed of one of the following: one or more white-space characters (space, tab, newline, vertical-tab or form-feed characters); an ordinary character (neither % nor a white-space character); or a conversion specification. Each conversion specification is introduced by the character % or the character sequence %n$ after which the following appear in sequence:

- An optional assignment-suppressing character *.
- An optional non-zero decimal integer that specifies the maximum field width.
- An optional size modifier h, l (ell) or L indicating the size of the receiving object. The conversion characters d, i and n must be preceded by h if the corresponding argument is a pointer to short int rather than a pointer to int, or by l (ell) if it is a pointer to long int. Similarly, the conversion characters o, u and x must be preceded by h if the corresponding argument is a pointer to unsigned short int rather than a pointer to unsigned int, or by l (ell) if it is a pointer to unsigned long int. Finally, the conversion characters e, f and g must be preceded by l (ell) if the corresponding argument is a pointer to double rather than a pointer to float, or by L if it is a pointer to long double. If an h, l (ell) or L appears with any other conversion character, the behaviour is undefined.
A conversion character that specifies the type of conversion to be applied. The valid conversion characters are described below.

The `fscanf()` functions execute each directive of the format in turn. If a directive fails, as detailed below, the function returns. Failures are described as input failures (due to the unavailability of input bytes) or matching failures (due to inappropriate input).

A directive composed of one or more white-space characters is executed by reading input until no more valid input can be read, or up to the first byte which is not a white-space character which remains unread.

A directive that is an ordinary character is executed as follows. The next byte is read from the input and compared with the byte that comprises the directive; if the comparison shows that they are not equivalent, the directive fails, and the differing and subsequent bytes remain unread.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each conversion character. A conversion specification is executed in the following steps:

1. Input white-space characters (as specified by `isspace()`) are skipped, unless the conversion specification includes a [, c, C or n conversion character.

2. An item is read from the input, unless the conversion specification includes an n conversion character. An input item is defined as the longest sequence of input bytes (up to any specified maximum field width, which may be measured in characters or bytes dependent on the conversion character) which is an initial subsequence of a matching sequence. The first byte, if any, after the input item remains unread. If the length of the input item is 0, the execution of the conversion specification fails; this condition is a matching failure, unless an error prevented input, in which case it is an input failure.

3. Except in the case of a % conversion character, the input item (or, in the case of a %n conversion specification, the count of input bytes) is converted to a type appropriate to the conversion character. If the input item is not a matching sequence, the execution of the conversion specification fails; this condition is a matching failure. Unless assignment suppression was indicated by a *, the result of the conversion is placed in the object pointed to by the first argument following the `format` argument that has not already received a conversion result if the conversion specification is introduced by %, or in the n'th argument if introduced by the character sequence `%n$`. If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behaviour is undefined.

The following conversion characters are valid:

- **d**: Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of `strtol()` with the value 10 for the base argument. In the absence of a size modifier, the corresponding argument must be a pointer to `int`.

- **i**: Matches an optionally signed integer, whose format is the same as expected for the subject sequence of `strtol()` with 0 for the base argument. In the absence of a size modifier, the corresponding argument must be a pointer to `int`.

- **o**: Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of `strtoul()` with the value 8 for the base argument. In the absence of a size modifier, the corresponding argument must be a pointer to `unsigned int`.

- **u**: Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of `strtoul()` with the value 10 for the base argument. In the absence of a size modifier, the corresponding argument must be a pointer to `unsigned int`. 

<table>
<thead>
<tr>
<th>Conversion Specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of <code>strtoul()</code> with the value 16 for the <code>base</code> argument. In the absence of a size modifier, the corresponding argument must be a pointer to <code>unsigned int</code>.</td>
</tr>
<tr>
<td>e, f, g</td>
<td>Matches an optionally signed floating-point number, whose format is the same as expected for the subject sequence of <code>strtol()</code>. In the absence of a size modifier, the corresponding argument must be a pointer to <code>float</code>.</td>
</tr>
<tr>
<td>s</td>
<td>Matches a sequence of bytes that are not white-space characters. The corresponding argument must be a pointer to the initial byte of an array of <code>char</code>, <code>signed char</code> or <code>unsigned char</code> large enough to accept the sequence and a terminating null character code, which will be added automatically.</td>
</tr>
<tr>
<td>[</td>
<td>Matches a non-empty sequence of bytes from a set of expected bytes (the <code>scanset</code>). The normal skip over white-space characters is suppressed in this case. The corresponding argument must be a pointer to the initial byte of an array of <code>char</code>, <code>signed char</code> or <code>unsigned char</code> large enough to accept the sequence and a terminating null byte, which will be added automatically. The conversion specification includes all subsequent bytes in the <code>format</code> string up to and including the matching right square bracket (]). The bytes between the square brackets (the <code>scanlist</code>) comprise the scanset, unless the byte after the left square bracket is a circumflex (ˆ), in which case the scanset contains all bytes that do not appear in the scanlist between the circumflex and the right square bracket. If the conversion specification begins with [ ] or [ˆ], the right square bracket is included in the scanlist and the next right square bracket is the matching right square bracket that ends the conversion specification; otherwise the first right square bracket is the one that ends the conversion specification.</td>
</tr>
</tbody>
</table>
conversion is undefined. The normal skip over white-space characters is suppressed in this case.

EX S Matches a sequence of characters that are not white space. The sequence is converted to a sequence of wide character codes in the same manner as `mbstowcs()` . The corresponding argument must be a pointer to the initial wide-character code of an array of `wchar_t` large enough to accept the sequence and a terminating null wide-character code, which will be added automatically. If the field width is specified, it denotes the maximum number of characters to accept.

EX % Matches a single %; no conversion or assignment occurs. The complete conversion specification must be %%. If a conversion specification is invalid, the behaviour is undefined.

The conversion characters E, G and X are also valid and behave the same as, respectively, e, g and x.

If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any bytes matching the current conversion specification (except for %n) have been read (other than leading white-space characters, where permitted), execution of the current conversion specification terminates with an input failure. Otherwise, unless execution of the current conversion specification is terminated with a matching failure, execution of the following conversion specification (if any) is terminated with an input failure.

Reaching the end of the string in `sscanf()` is equivalent to encountering end-of-file for `fscanf()`.

If conversion terminates on a conflicting input, the offending input is left unread in the input. Any trailing white space (including newline characters) is left unread unless matched by a conversion specification. The success of literal matches and suppressed assignments is only directly determinable via the %n conversion specification.

The `fscanf()` and `scanf()` functions may mark the `st_atime` field of the file associated with `stream` for update. The `st_atime` field will be marked for update by the first successful execution of `fgetc()`, `fgets()`, `fread()`, `getc()`, `getchar()`, `gets()`, `fscanf()` or `scanf()` using `stream` that returns data not supplied by a prior call to `ungetc()`.

RETURN VALUE

Upon successful completion, these functions return the number of successfully matched and assigned input items; this number can be 0 in the event of an early matching failure. If the input ends before the first matching failure or conversion, EOF is returned. If a read error occurs the error indicator for the stream is set, EOF is returned, and `errno` is set to indicate the error.

ERRORS

For the conditions under which the `fscanf()` functions will fail and may fail, refer to `fgetc()` or `fgetwc()`.

In addition, `fscanf()` may fail if:

EX [EILSEQ] Input byte sequence does not form a valid character.

EX [EINVAL] There are insufficient arguments.
EXAMPLES
The call:

```c
int i, n; float x; char name[50];
n = scanf("%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E-1 Hamster
```

will assign to `n` the value 3, to `i` the value 25, to `x` the value 5.432, and `name` will contain the string Hamster.

The call:

```c
int i; float x; char name[50];
(void) scanf("%2d%f%*d %\[0123456789\]", &i, &x, name);
```

with input:

```
56789 0123 56a72
```

will assign 56 to `i`, 789.0 to `x`, skip 0123, and place the string 56\0 in `name`. The next call to `getchar()` will return the character a.

APPLICATION USAGE
If the application calling `fprintf()` has any objects of type `wchar_t`, it must also include either `<sys/types.h>` or `<stddef.h>` to have `wchar_t` defined.

The `fscanf()` function may recognise character string representations for infinity and NaN (a symbolic entity encoded in floating-point format) to support the ANSI/IEEE Std 754:1985 standard.

In format strings containing the % form of conversion specifications, each argument in the argument list is used exactly once.

SEE ALSO
`getc()`, `printf()`, `setlocale()`, `strtol()`, `strtoul()`, `<langinfo.h>`, `<stdio.h>`, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated for alignment with the ISO C standard:

- The type of the argument `format` for all functions, and the type of argument `s` for `sscanf()`, are changed from `char *` to `const char *`.

- The description is updated in various places to align more closely with the text of the ISO C standard. In particular, this issue fully defines the L conversion character, allows for the support of multi-byte coded character sets (although these are not mandated by X/Open), and fills in a number of gaps in the definition (for example, by defining termination conditions for `sscanf()`).

- Following an ANSI interpretation, the effect of conversion specifications that consume no input is better defined, and is no longer marked as an extension.
The following change is incorporated for alignment with the MSE working draft.

- The C and S conversion characters are added, indicating a pointer in the argument list to the initial wide-character code of an array large enough to accept the input sequence.

Other changes are incorporated as follows:

- Use of the terms “byte” and “character” is rationalised to make it clear when single-byte and multi-byte values can be used. Similarly, use of the terms “conversion specification” and “conversion character” is now more precise.

- Various errors are corrected. For example, the description of the d conversion character contained an erroneous reference to \texttt{strtol}() in Issue 3. This is replaced in this issue by reference to \texttt{strtol}().

- The \texttt{DESCRIPTION} section is updated in a number of places to indicate further implications of the \texttt{%n$} form of a conversion. All references to this functionality, which is not specified in the ISO C standard, are marked as extensions.

- The \texttt{ERRORS} section is changed to refer to the entries for \texttt{fgetc()} and \texttt{fgetwc}(); the [EINVAL] error is marked as an extension; and the [EILSEQ] error is added and marked as an extension.

- The detailed description of this function including the \texttt{CHANGE HISTORY} section for \texttt{scanf()} is provided here instead of under \texttt{scanf}().

- The \texttt{APPLICATION USAGE} section is amended to record the need for \texttt{<sys/types.h>} or \texttt{<stddef.h>} if type \texttt{wchar_t} is required.
NAME
fseek — reposition a file-position indicator in a stream

SYNOPSIS
#include <stdio.h>

int fseek(FILE *stream, long int offset, int whence);

DESCRIPTION
The fseek() function sets the file-position indicator for the stream pointed to by stream.

The new position, measured in bytes from the beginning of the file, is obtained by adding offset to the position specified by whence. The specified point is the beginning of the file for SEEK_SET, the current value of the file-position indicator for SEEK_CUR, or end-of-file for SEEK_END.

If the stream is to be used with wide character input/output functions, offset must either be 0 or a value returned by an earlier call to ftell() on the same stream and whence must be SEEK_SET.

A successful call to fseek() clears the end-of-file indicator for the stream and undoes any effects of ungetc() and ungetwc() on the same stream. After an fseek() call, the next operation on an update stream may be either input or output.

If the most recent operation, other than ftell(), on a given stream is fflush(), the file offset in the underlying open file description will be adjusted to reflect the location specified by fseek().

The fseek() function allows the file-position indicator to be set beyond the end of existing data in the file. If data is later written at this point, subsequent reads of data in the gap will return bytes with the value 0 until data is actually written into the gap.

The behaviour of fseek() on devices which are incapable of seeking is implementation-dependent. The value of the file offset associated with such a device is undefined.

If the stream is writable and buffered data had not been written to the underlying file, fseek() will cause the unwritten data to be written to the file and mark the st_ctime and st_mtime fields of the file for update.

RETURN VALUE
EX The fseek() function returns 0 if it succeeds; otherwise it returns −1 and sets errno to indicate the error.

ERRORS
The fseek() function will fail if, either the stream is unbuffered or the stream’s buffer needed to be flushed, and the call to fseek() causes an underlying lseek() or write() to be invoked:

[EAGAIN] The O_NONBLOCK flag is set for the file descriptor and the process would be delayed in the write operation.

[EBADFD] The file descriptor underlying the stream file is not open for writing or the stream’s buffer needed to be flushed and the file is not open.

[EFBIG] An attempt was made to write a file that exceeds the maximum file size or the process’ file size limit.

[EINTR] The write operation was terminated due to the receipt of a signal, and no data was transferred.

[EINVAL] The whence argument is invalid. The resulting file-position indicator would be set to a negative value.

[EIO] A physical I/O error has occurred, or the process is a member of a background process group attempting to perform a write() to its controlling
fseek() BASE System Interfaces

terminal, TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU and the process group of the process is orphaned. This error may also be returned under implementation-dependent conditions.

[ENOSPC] There was no free space remaining on the device containing the file.

[EPIPE] (a) The file descriptor underlying stream is associated with a pipe or FIFO.

(b) An attempt was made to write to a pipe or FIFO that is not open for reading by any process; a SIGPIPE signal will also be sent to the process.

EX [ENXIO] A request was made of a non-existent device, or the request was outside the capabilities of the device.

APPLICATION USAGE
In a locale with state-dependent encoding, whether fseek() restores the stream’s shift state is implementation-dependent.

SEE ALSO
fopen(), ftell(), getrlimit(), rewind(), ulimit(), ungetc(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:

• The type of argument offset is now defined in full as long int instead of long.

The following change is incorporated for alignment with the FIPS requirements:

• The [EINTR] error is no longer an indication that the implementation does not report partial transfers.

Other changes are incorporated as follows:

• In the DESCRIPTION section, the words ‘‘The seek() function does not, by itself, extend the size of a file’’ are deleted.

• In the RETURN VALUE section, the value −1 is marked as an extension. This is because the ISO POSIX-1 standard only requires that a non-zero value is returned.

• In the ERRORS section, text is added to indicate that error returns will only be generated when either the stream is unbuffered, or if the stream buffer needs to be flushed.

• The ‘‘will fail’’ and ‘‘may fail’’ parts of the ERRORS section are revised for consistency with lseek() and write().

• Text associated with the [EIO] error is expanded and the [ENXIO] error is added.

• Text is added to explain how fseek() is used with wide character input/output; this is marked as a WP extension.

• The [EFBIG] error is marked to show extensions.

• The APPLICATION USAGE section is added.

Issue 4, Version 2
In the ERRORS section, the description of [EIO] is updated to include the case where a physical I/O error occurs.
NAME
fsetpos — set current file position

SYNOPSIS
#include <stdio.h>
int fsetpos(FILE *stream, const fpos_t *pos);

DESCRIPTION
The fsetpos() function sets the file position indicator for the stream pointed to by stream
according to the value of the object pointed to by pos, which must be a value obtained from an
earlier call to fgetpos() on the same stream.

A successful call to fsetpos() function clears the end-of-file indicator for the stream and undoes
any effects of ungetc() on the same stream. After an fsetpos() call, the next operation on an
update stream may be either input or output.

RETURN VALUE
The fsetpos() function returns 0 if it succeeds; otherwise it returns a non-zero value and sets errno
to indicate the error.

ERRORS
The fsetpos() function may fail if:

EX  [EBADF] The file descriptor underlying stream is not valid.
    [ESPIPE] The file descriptor underlying stream is associated with a pipe or FIFO.

SEE ALSO
fopen(), ftell(), rewind(), ungetc(), <stdio.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the ISO C standard.
NAME
fstat — get file status

SYNOPSIS

#include <sys/types.h>
#include <sys/stat.h>

int fstat(int fildes, struct stat *buf);

DESCRIPTION

The fstat() function obtains information about an open file associated with the file descriptor fildes, and writes it to the area pointed to by buf.

The buf argument is a pointer to a stat structure, as defined in <sys/stat.h>, into which information is placed concerning the file.

The structure members st_mode, st_ino, st_dev, st_uid, st_gid, st_atime, st_ctime and st_mtime will have meaningful values for all file types defined in this document. The value of the member st_nlink will be set to the number of links to the file.

An implementation that provides additional or alternative file access control mechanisms may, under implementation-dependent conditions, cause fstat() to fail.

The fstat() function updates any time-related fields as described in File Times Update (see the XBD specification, Chapter 4, Character Set), before writing into the stat structure.

RETURN VALUE

Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS

The fstat() function will fail if:

[EBADF] The fildes argument is not a valid file descriptor.

[EINVAL] An I/O error occurred while reading from the file system.

The fstat() function may fail if:

[EINVAL] One of the values is too large to store into the structure pointed to by the buf argument.

SEE ALSO

lstat(), stat(), <sys/stat.h>, <sys/types.h>.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated in the DESCRIPTION section for alignment with the ISO POSIX-1 standard:

• A paragraph defining the contents of stat structure members is added.

• The words “extended security controls” are replaced by “additional or alternative file access control mechanisms”.

210 X/Open CAE Specification (1994)
Another change is incorporated as follows:

- The header `<sys/types.h>` is now marked as optional (OH); this header need not be included on XSI-conformant systems.

**Issue 4, Version 2**

The **ERRORS** section is updated for X/Open UNIX conformance as follows:

- The `[EIO]` error is added as a mandatory error indicated the occurrence of an I/O error.
- The `[EOVERFLOW]` error is added as an optional error indicating that one of the values is too large to store in the area pointed to by `buf`. 
NAME
fstatvfs, statvfs — get file system information

SYNOPSIS
#include <sys/statvfs.h>

int fstatvfs(int fildes, struct statvfs *buf);
int statvfs(const char *path, struct statvfs *buf);

DESCRIPTION
The fstatvfs() function obtains information about the file system containing the file referenced by fildes.

The following flags can be returned in the f_flag member:
- ST_RDONLY     read-only file system
- ST_NOSUID     setuid/setgid bits ignored by exec

The statvfs() function obtains descriptive information about the file system containing the file named by path.

For both functions, the buf argument is a pointer to a statvfs structure that will be filled. Read, write, or execute permission of the named file is not required, but all directories listed in the pathname leading to the file must be searchable.

RETURN VALUE
Upon successful completion, statvfs() returns 0. Otherwise, it returns -1 and sets errno to indicate the error.

ERRORS
The fstatvfs() and statvfs() functions will fail if:
- [EIO] An I/O error occurred while reading the file system.
- [EINTR] A signal was caught during execution of the function.

The fstatvfs() function will fail if:
- [EBADF] The fildes argument is not an open file descriptor.

The statvfs() function will fail if:
- [EACCES] Search permission is denied on a component of the path prefix.
- [ELOOP] Too many symbolic links were encountered in resolving path.
- [ENAMETOOLONG] The length of a pathname exceeds PATH_MAX, or a pathname component is longer than NAME_MAX.
- [ENOENT] A component of path does not name an existing file or path is an empty string.
- [ENOTDIR] A component of the path prefix of path is not a directory.

The statvfs() function may fail if:
- [ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds PATH_MAX.

APPLICATION USAGE
It is unspecified whether all members of the statvfs structure have meaningful values on all file systems.
SEE ALSO

    chmod(), chown(), creat(), dup(), exec, fcntl(), link(), mknod(), open(), pipe(), read(), time(),
    unlink(), ustat(), utime(), write(), <sys/statvfs.h>.

CHANGE HISTORY

    First released in Issue 4, Version 2.
NAME

fsync — synchronise changes to a file

SYNOPSIS

```c
#include <unistd.h>
int fsync(int fildes);
```

DESCRIPTION

The `fsync()` function causes all modified data and attributes of the file referred to by `fildes` to be delivered to the underlying hardware.

RETURN VALUE

Upon successful completion, `fsync()` returns 0. Otherwise, −1 is returned and `errno` is set to indicate the error.

ERRORS

The `fsync()` function will fail if:

- `[EBADF]` The `fildes` argument is not a valid descriptor.
- `[EINTR]` The `fsync()` function was interrupted by a signal.
- `[EINVAL]` The `fildes` argument does not refer to a file on which this operation is possible.
- `[EIO]` An I/O error occurred while reading from or writing to the file system.

APPLICATION USAGE

The `fsync()` function should be used by programs which require modifications to a file to be completed before continuing; for example, a program which contains a simple transaction facility might use it to ensure that all modifications to a file or files caused by a transaction are recorded.

SEE ALSO

`sync()`, `<unistd.h>`.

CHANGE HISTORY

First released in Issue 3.

Issue 4

The following changes are incorporated in this issue:

- The header `<unistd.h>` is added to the SYNOPSIS.
- In the APPLICATION USAGE section, the words “require a file to be in a known state” are replaced by “require modifications to a file to be completed before continuing”.

214 X/Open CAE Specification (1994)
NAME

ftell — return a file offset in a stream

SYNOPSIS

#include <stdio.h>

long int ftell(FILE *stream);

DESCRIPTION

The ftell() function obtains the current value of the file-position indicator for the stream pointed to by stream.

RETURN VALUE

Upon successful completion, ftell() returns the current value of the file-position indicator for the stream measured in bytes from the beginning of the file.

Otherwise, ftell() returns −1L and sets errno to indicate the error.

ERRORS

The ftell() function will fail if:

[EBADF] The file descriptor underlying stream is not an open file descriptor.

[ESPIPE] The file descriptor underlying stream is associated with a pipe or FIFO.

SEE ALSO

fopen(), fseek(), lseek(), <stdio.h>.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following change is incorporated for alignment with the ISO C standard:

• The function return value is now defined in full as long int. It was previously defined as long.
NAME
ftime — get date and time

SYNOPSIS

UX

```c
#include <sys/timeb.h>

int ftime(struct timeb *tp);
```

DESCRIPTION
The `ftime()` function sets the `time` and `millitm` members of the `timeb` structure pointed to by `tp` to contain the seconds and milliseconds portions, respectively, of the current time in seconds since 00:00:00 UTC (Coordinated Universal Time), January 1, 1970. The contents of the `timezone` and `dstflag` members of `tp` after a call to `ftime()` are unspecified.

RETURN VALUE
Upon successful completion, the `ftime()` function returns 0. Otherwise −1 is returned.

ERRORS
No errors are defined.

APPLICATION USAGE
For portability to implementations conforming to earlier versions of this document, `time()` is preferred over this function.

The millisecond value usually has a granularity greater than one due to the resolution of the system clock. Depending on any granularity (particularly a granularity of one) renders code non-portable.

SEE ALSO
`ctime()`, `gettimeofday()`, `time()`, `<sys/timeb.h>`.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
ftok — generate an IPC key

SYNOPSIS
UX
#include <sys/ipc.h>
key_t ftok(const char *path, int id);

DESCRIPTION
The ftok() function returns a key based on path and id that is usable in subsequent calls to msgget(), semget() and shmget(). The path argument must be the pathname of an existing file that the process is able to stat().

The ftok() function will return the same key value for all paths that name the same file, when called with the same id value, and will return different key values when called with different id values or with paths that name different files existing on the same file system at the same time. It is unspecified whether ftok() returns the same key value when called again after the file named by path is removed and recreated with the same name.

Only the low order 8-bits of id are significant. The behaviour of ftok() is unspecified if these bits are 0.

RETURN VALUE
Upon successful completion, ftok() returns a key. Otherwise, ftok() returns (key_t)-1 and sets errno to indicate the error.

ERRORS
The ftok() function will fail if:

[EACCES] Search permission is denied for a component of the path prefix.
[ELOOP] Too many symbolic links were encountered in resolving path.
[ENAMETOOLONG] The length of the path argument exceeds [PATH_MAX] or a pathname component is longer than [NAME_MAX].
[ENOENT] A component of path does not name an existing file or path is an empty string.
[ENOTDIR] A component of the path prefix is not a directory.

The ftok() function may fail if:

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds [PATH_MAX].

APPLICATION USAGE
For maximum portability, id should be a single-byte character.

SEE ALSO
msgget(), semget(), shmget(), <sys/ipc.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
ftruncate()  X/Open UNIX  System Interfaces

NAME
ftruncate, truncate — truncate a file to a specified length

SYNOPSIS
UX
#include <unistd.h>

int ftruncate(int fildes, off_t length);

int truncate(const char *path, off_t length);

DESCRIPTION
The ftruncate() function causes the regular file referenced by fildes to have a size of length bytes.
The truncate() function causes the regular file named by path to have a size of length bytes.
The effect of ftruncate() and truncate() on other types of files is unspecified. If the file previously was larger than length, the extra data is lost. If it was previously shorter than length, bytes between the old and new lengths are read as zeroes. With ftruncate(), the file must be open for writing; for truncate(), the process must have write permission for the file.

If the request would cause the file size to exceed the soft file size limit for the process, the request will fail and the implementation will generate the SIGXFSZ signal for the process.

These functions do not modify the file offset for any open file descriptions associated with the file. On successful completion, if the file size is changed, these functions will mark for update the st_ctime and st_mtime fields of the file, and if the file is a regular file, the S_ISUID and S_ISGID bits of the file mode may be cleared.

RETURN VALUE
Upon successful completion, ftruncate() and truncate() returns 0. Otherwise a −1 is returned, and errno is set to indicate the error.

ERRORS
The ftruncate() and truncate() functions will fail if:

[EINTR] A signal was caught during execution.

[EINVAL] The length argument was less than 0.

[EFBIG] or [EINVAL] The length argument was greater than the maximum file size.

[EIO] An I/O error occurred while reading from or writing to a file system.

The ftruncate() function will fail if:

[EBADF] or [EINVAL] The fildes argument is not a file descriptor open for writing.

[EINVAL] The fildes argument references a file that was opened without write permission.

The truncate() function will fail if:

[EACCES] A component of the path prefix denies search permission, or write permission is denied on the file.

[EISDIR] The named file is a directory.

[ELOOP] Too many symbolic links were encountered in resolving path.

[ENAMETOOLONG] The length of the specified pathname exceeds PATH_MAX bytes, or the length
The `ftruncate()` function may fail if:

- **[ENOENT]** A component of `path` does not name an existing file or `path` is an empty string.
- **[ENOTDIR]** A component of the path prefix of `path` is not a directory.
- **[EROFS]** The named file resides on a read-only file system.

Pathname resolution of a symbolic link produced an intermediate result whose length exceeds `{PATH_MAX}`.

**SEE ALSO**

`open()`, `<unistd.h>`.

**CHANGE HISTORY**

First released in Issue 4, Version 2.
NAME

ftw — traverse (walk) a file tree

SYNOPSIS

```
#include <ftw.h>

int ftw(const char * path,
        int (*fn)(const char *, const struct stat * ptr, int flag),
        int ndirs);
```

DESCRIPTION

The `ftw()` function recursively descends the directory hierarchy rooted in `path`. For each object in the hierarchy, `ftw()` calls the function pointed to by `fn`, passing it a pointer to a null-terminated character string containing the name of the object, a pointer to a `stat` structure containing information about the object, and an integer. Possible values of the integer, defined in the `<ftw.h>` header, are:

- FTW_D for a directory
- FTW_DNR for a directory that cannot be read
- FTW_F for a file
- FTW_SL for a symbolic link (but see also FTW_NS below)
- FTW_NS for an object other than a symbolic link on which `stat()` could not successfully be executed. If the object is a symbolic link and `stat()` failed, it is unspecified whether `ftw()` passes FTW_SL or FTW_NS to the user-supplied function.

If the integer is FTW_DNR, descendants of that directory will not be processed. If the integer is FTW_NS, the `stat` structure will contain undefined values. An example of an object that would cause FTW_NS to be passed to the function pointed to by `fn` would be a file in a directory with read but without execute (search) permission.

The `ftw()` function visits a directory before visiting any of its descendants.

- The `ftw()` function uses at most one file descriptor for each level in the tree.

The argument `ndirs` should be in the range of 1 to `{OPEN_MAX}`.

The tree traversal continues until the tree is exhausted, an invocation of `fn` returns a non-zero value, or some error, other than [EACCES], is detected within `ftw()`.

The `ndirs` argument specifies the maximum number of directory streams or file descriptors or both available for use by `ftw()` while traversing the tree. When `ftw()` returns it closes any directory streams and file descriptors it uses not counting any opened by the application-supplied `fn()` function.

RETURN VALUE

If the tree is exhausted, `ftw()` returns 0. If the function pointed to by `fn` returns a non-zero value, `ftw()` stops its tree traversal and returns whatever value was returned by the function pointed to by `fn()`. If `ftw()` detects an error, it returns −1 and sets `errno` to indicate the error.

- If `ftw()` encounters an error other than [EACCES] (see FTW_DNR and FTW_NS above), it returns −1 and `errno` is set to indicate the error. The external variable `errno` may contain any error value that is possible when a directory is opened or when one of the `stat` functions is executed on a directory or file.
ERRORS

The \texttt{ftw()} function will fail if:

- \texttt{[EACCES]} Search permission is denied for any component of \texttt{path} or read permission is denied for \texttt{path}.
- \texttt{[ELOOP]} Too many symbolic links were encountered.
- \texttt{[ENAMETOOLONG]} The length of the \texttt{path} exceeds \{PATH\_MAX\}, or a pathname component is longer than \{NAME\_MAX\}.
- \texttt{[ENOENT]} A component of \texttt{path} does not name an existing file or \texttt{path} is an empty string.
- \texttt{[ENOTDIR]} A component of \texttt{path} is not a directory.

The \texttt{ftw()} function may fail if:

- \texttt{[EINVAL]} The value of the \texttt{ndirs} argument is invalid.

UX

- \texttt{[ENAMETOOLONG]} Pathname resolution of a symbolic link produced an intermediate result whose length exceeds \{PATH\_MAX\}.

In addition, if the function pointed to by \texttt{fn} encounters system errors, \texttt{errno} may be set accordingly.

APPLICATION USAGE

Because \texttt{ftw()} is recursive, it is possible for it to terminate with a memory fault when applied to very deep file structures.

The \texttt{ftw()} function uses \texttt{malloc()} to allocate dynamic storage during its operation. If \texttt{ftw()} is forcibly terminated, such as by \texttt{longjmp()} or \texttt{siglongjmp()} being executed by the function pointed to by \texttt{fn} or an interrupt routine, \texttt{ftw()} will not have a chance to free that storage, so it will remain permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have the function pointed to by \texttt{fn} return a non-zero value at its next invocation.

SEE ALSO

\texttt{longjmp()}, \texttt{lstat()}, \texttt{malloc()}, \texttt{opendir()}, \texttt{siglongjmp()}, \texttt{stat()}, \texttt{<ftw.h>}, \texttt{<sys/stat.h>}.  

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following change is incorporated for alignment with the FIPS requirements:

- In the \textbf{ERRORS} section, the condition whereby \[ENAMETOOLONG\] will be returned if a pathname component is larger that \{NAME\_MAX\} is now defined as mandatory and marked as an extension.

Other changes are incorporated as follows:

- The type of argument \texttt{path} is changed from \texttt{char *} to \texttt{const char *}. The argument list for \texttt{fn()} has also been defined.

- In the \textbf{DESCRIPTION} section, the words “other than \[EACCES]\” are added to the paragraph describing termination conditions for tree traversal.
Issue 4, Version 2
The following changes are incorporated for X/OPEN UNIX conformance:

- The DESCRIPTION is updated to describe the use of the FTW_SL and FTW_NS values for a symbolic link.
- The DESCRIPTION states that ftw() uses at most one file descriptor for each level in the tree.
- The DESCRIPTION constrains ndirs to the range from 1 to OPEN_MAX.
- The RETURN VALUE section is updated to describe the case where ftw() encounters an error other than EACCES.
- In the ERRORS section, a second ENAMETOOLONG condition is defined that may report excessive length of an intermediate result of pathname resolution of a symbolic link.
NAME
fwrite — binary output

SYNOPSIS
#include <stdio.h>

size_t fwrite(const void *ptr, size_t size, size_t nitems,
               FILE *stream);

DESCRIPTION
The fwrite() function writes, from the array pointed to by ptr, up to nitems members whose size
is specified by size, to the stream pointed to by stream. The file-position indicator for the stream
(if defined) is advanced by the number of bytes successfully written. If an error occurs, the
resulting value of the file-position indicator for the stream is indeterminate.

The st_ctime and st_mtime fields of the file will be marked for update between the successful
execution of fwrite() and the next successful completion of a call to fflush() or fclose() on the
same stream or a call to exit() or abort().

RETURN VALUE
The fwrite() function returns the number of members successfully written, which may be less
than nitems if a write error is encountered. If size or nitems is 0, fwrite() returns 0 and the state of
the stream remains unchanged. Otherwise, if a write error occurs, the error indicator for the
stream is set and errno is set to indicate the error.

ERRORS
Refer to fputc().

SEE ALSO
ferror(), fopen(), printf(), putc(), puts(), write(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
• The type of argument ptr is changed from void* to const void*.

Another change is incorporated as follows:
• In the DESCRIPTION section, the text is changed to make it clear that the function advances
  the file-position indicator by the number of bytes successfully written rather than the number
  of characters, which could include multi-byte sequences.
gamma()  BASE  System Interfaces

NAME
  gamma, signgam — log gamma function (TO BE WITHDRAWN)

SYNOPSIS
  EX  #include <math.h>
      double gamma(double x);
      extern int signgam;

DESCRIPTION
  The gamma() function performs identically to lgamma(), including the use of signgam.

APPLICATION USAGE
  This interface is functionally equivalent to lgamma() and so it is marked to be withdrawn.

CHANGE HISTORY
  First released in Issue 1.
  Derived from Issue 1 of the SVID.

Issue 4
  The following changes are incorporated in this issue:
    • This interface is marked TO BE WITHDRAWN, as it is functionally equivalent to lgamma().
    • The DESCRIPTION section is changed to refer to lgamma().
    • The APPLICATION USAGE section is added.
NAME
gcvt — convert floating-point number to string

SYNOPSIS

```
 UX
 #include <stdlib.h>

 char *gcvt(double value, int ndigit, char *buf);
```

DESCRIPTION

Refer to `ecvt()`.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
getc — get byte from a stream

SYNOPSIS
#include <stdio.h>
int getc(FILE *stream);

DESCRIPTION
The getc() function is equivalent to fgetc(), except that if it is implemented as a macro it may evaluate stream more than once, so the argument should never be an expression with side effects.

RETURN VALUE
Refer to fgetc().

ERRORS
Refer to fgetc().

APPLICATION USAGE
If the integer value returned by getc() is stored into a variable of type char and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a variable of type char on widening to integer is implementation-dependent.

Because it may be implemented as a macro, getc() may treat incorrectly a stream argument with side effects. In particular, getc(*f++) may not work as expected. Therefore, use of this function is not recommended in such situations; fgetc() should be used instead.

SEE ALSO
fgetc(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

- The words “a character variable” are replaced by “a variable of type char”, to emphasise the fact that this interface deals with byte values.

- The APPLICATION USAGE section now states that the use of this function is not recommended.
NAME

getchar — get byte from stdin stream

SYNOPSIS

#include <stdio.h>

int getchar(void);

DESCRIPTION

The getchar() function is equivalent to getc(stdin).

RETURN VALUE

Refer to fgetc().

ERRORS

Refer to fgetc().

APPLICATION USAGE

If the integer value returned by getchar() is stored into a variable of type char and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a variable of type char on widening to integer is implementation-dependent.

SEE ALSO

getc(), <stdio.h>.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following change is incorporated for alignment with the ISO C standard:

• The argument list is explicitly defined as void.

Another change is incorporated as follows:

• The words “a character variable” are replaced by “a variable of type char”, to emphasise the fact that this interface deals in byte values.
NAME
getcontext, setcontext — get and set current user context

SYNOPSIS
UX
#include <ucontext.h>

int getcontext(ucontext_t *ucp);
int setcontext(const ucontext_t *ucp);

DESCRIPTION
The getcontext() function initialises the structure pointed to by ucp to the current user context of
the calling process. The ucontext_t type that ucp points to defines the user context and includes
the contents of the calling process' machine registers, the signal mask, and the current execution
stack.

The setcontext() function restores the user context pointed to by ucp. A successful call to
setcontext() does not return; program execution resumes at the point specified by the ucp
argument passed to setcontext(). The ucp argument should be created either by a prior call to
getcontext(), or by being passed as an argument to a signal handler. If the ucp argument was
created with getcontext(), program execution continues as if the corresponding call of
getcontext() had just returned. If the ucp argument was created with makecontext(), program
execution continues with the function passed to makecontext(). When that function returns, the
process continues as if after a call to setcontext() with the ucp argument that was input to
makecontext(). If the ucp argument was passed to a signal handler, program execution continues
with the program instruction following the instruction interrupted by the signal. If the uc_link
member of the ucontext_t structure pointed to by the ucp argument is equal to 0, then this
context is the main context, and the process will exit when this context returns. The effects of
passing a ucp argument obtained from any other source are unspecified.

RETURN VALUE
On successful completion, setcontext() does not return and getcontext() returns 0. Otherwise, a
value of −1 is returned.

ERRORS
No errors are defined.

APPLICATION USAGE
When a signal handler is executed, the current user context is saved and a new context is
created. If the process leaves the signal handler via longjmp(), then it is unspecified whether the
context at the time of the corresponding setjmp() call is restored and thus whether future calls to
getcontext() will provide an accurate representation of the current context, since the context
restored by longjmp() may not contain all the information that setcontext() requires. Signal
handlers should use siglongjmp() or setcontext() instead.

Portable applications should not modify or access the uc_mcontext member of ucontext_t. A
portable application cannot assume that context includes any process-wide static data, possibly
including errno. Users manipulating contexts should take care to handle these explicitly when
required.

SEE ALSO
bsd_signal(), makecontext(), setjmp(), sigaction(), sigaltstack(), sigprocmask(), sigsetjmp(),
<ucontext.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
getcwd — get pathname of current working directory

SYNOPSIS
#include <unistd.h>
char *getcwd(char *buf, size_t size);

DESCRIPTION
The getcwd() function places an absolute pathname of the current working directory in the array
pointed to by buf, and returns buf. The size argument is the size in bytes of the character array
pointed to by the buf argument. If buf is a null pointer, the behaviour of getcwd() is undefined.

RETURN VALUE
Upon successful completion, getcwd() returns the buf argument. Otherwise, getcwd() returns a
null pointer and sets errno to indicate the error. The contents of the array pointed to by buf is
then undefined.

ERRORS
The getcwd() function will fail if:
[EINVAL] The size argument is 0.
[ERANGE] The size argument is greater than 0, but is smaller than the length of the
pathname +1.

The getcwd() function may fail if:
[EACCES] Read or search permission was denied for a component of the pathname.

[ENOMEM] Insufficient storage space is available.

APPLICATION USAGE
If buf is a null pointer, on some implementations, getcwd() will obtain size bytes of space using
malloc(). In this case, the pointer returned by getcwd() may be used as the argument in a
subsequent call to free(). Invoking getcwd() with buf as a null pointer is not recommended.

SEE ALSO
malloc(), <unistd.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:
• The DESCRIPTION section is changed to indicate that the effects of passing a null pointer in
buf are undefined.

Other changes are incorporated as follows:
• The header <unistd.h> is added to the SYNOPSIS section.
• The [ENOMEM] error is marked as an extension.
• The words “as this functionality may be subject to withdrawal” have been deleted from the
end of the last sentence in the APPLICATION USAGE section.
NAME
getdate — convert user format date and time

SYNOPSIS

UX

```
#include <time.h>

struct tm *getdate(const char *string);
```

DESCRIPTION

The `getdate()` function converts a string representation of a date or time into a broken-down time.

Templates are used to parse and interpret the input string. The templates are contained in a text file identified by the environment variable `DATEMSK`. The `DATEMSK` variable should be set to indicate the full pathname of the file that contains the templates. The first line in the template that matches the input specification is used for interpretation and conversion into the internal time format.

The following field descriptors are supported:

- `%%` same as `%`
- `%a` abbreviated weekday name
- `%A` full weekday name
- `%b` abbreviated month name
- `%B` full month name
- `%c` locale's appropriate date and time representation
- `%d` day of month (01-31; the leading 0 is optional)
- `%D` date as `%m/%d/%y`
- `%e` same as `%d`
- `%h` abbreviated month name
- `%H` hour (00-23)
- `%I` hour (01-12)
- `%m` month number (01-12)
- `%M` minute (00-59)
- `%n` same as new line
- `%p` locale’s equivalent of either AM or PM
- `%r` The locale’s appropriate representation of time in AM and PM notation. In the POSIX locale, this is equivalent to `%I:%M:%S %p`
- `%R` The locale’s appropriate representation of time. In the POSIX locale, this is equivalent to `%H:%M`
- `%S` seconds (00-61). Leap seconds are allowed but are not predictable through use of algorithms.
- `%t` same as tab
The match between the template and input specification performed by `getdate()` is case insensitive.

The month and weekday names can consist of any combination of upper and lower case letters. The process can request that the input date or time specification be in a specific language by setting the LC_TIME category (see `setlocale()`).

Leading 0's are not necessary for the descriptors that allow leading 0's. However, at most two digits are allowed for those descriptors, including leading 0's. Extra whitespace in either the template file or in `string` is ignored.

The field descriptors `%c`, `%x`, and `%X` will not be supported if they include unsupported field descriptors.

The following rules apply for converting the input specification into the internal format:

- If `%Z` is being scanned, then `getdate()` initialises the broken-down time to be the current time in the scanned time zone. Otherwise it initialises the broken-down time based on the current local time as if `localtime()` had been called.
- If only the weekday is given, today is assumed if the given day is equal to the current day and next week if it is less,
- If only the month is given, the current month is assumed if the given month is equal to the current month and next year if it is less and no year is given (the first day of month is assumed if no day is given),
- If no hour, minute and second are given the current hour, minute and second are assumed,
- If no date is given, today is assumed if the given hour is greater than the current hour and tomorrow is assumed if it is less.

The external variable or macro `getdate_err` is used by `getdate()` to return error values.

**RETURN VALUE**

Upon successful completion, `getdate()` returns a pointer to a `struct tm`. Otherwise, it returns a null pointer and `getdate_err` is set to indicate the error.

**ERRORS**

The `getdate()` function will fail in the following cases, setting `getdate_err` to the value shown in the list below. Any changes to `errno` are unspecified.

1. The `DATEMSK` environment variable is null or undefined.
2. The template file cannot be opened for reading.
3 Failed to get file status information.
4 The template file is not a regular file.
5 An error is encountered while reading the template file.
6 Memory allocation failed (not enough memory available).
7 There is no line in the template that matches the input.
8 Invalid input specification. For example, February 31; or a time is specified that can not be represented in a `time_t` (representing the time in seconds since 00:00:00 UTC, January 1, 1970).

**EXAMPLE**

**Example 1:**

The following example shows the possible contents of a template:

```plaintext
%m
%A %B %d, %Y, %H:%M:%S
%A %B %m/%d/%y %I %p
%d,%m,%Y %H:%M
at %A the %dst of %B in %Y
run job at %I %p,%B %dnd
%A den %d. %B %Y %H.%M Uhr
```

**Example 2:**

The following are examples of valid input specifications for the template in Example 1:

```plaintext
getdate("10/1/87 4 PM");
getdate("Friday");
getdate("Friday September 18, 1987, 10:30:30");
getdate("24,9,1986 10:30");
getdate("at monday the 1st of december in 1986");
getdate("run job at 3 PM, december 2nd");
```

If the LC_TIME category is set to a German locale that includes `freitag` as a weekday name and `oktober` as a month name, the following would be valid:

```plaintext
getdate("freitag den 10. oktober 1986 10.30 Uhr");
```

**Example 3:**

The following examples shows how local date and time specification can be defined in the template.

<table>
<thead>
<tr>
<th>Invocation</th>
<th>Line in Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>getdate(&quot;11/27/86&quot;)</td>
<td>%m/%d/%y</td>
</tr>
<tr>
<td>getdate(&quot;27.11.86&quot;)</td>
<td>%d.%m.%y</td>
</tr>
<tr>
<td>getdate(&quot;86-11-27&quot;)</td>
<td>%y-%m-%d</td>
</tr>
<tr>
<td>getdate(&quot;Friday 12:00:00&quot;)</td>
<td>%A %H:%M:%S</td>
</tr>
</tbody>
</table>
Example 4:
The following examples help to illustrate the above rules assuming that the current date is Mon Sep 22 12:19:47 EDT 1986 and the LC_TIME category is set to the default "C" locale.

<table>
<thead>
<tr>
<th>Input</th>
<th>Line in Template</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon</td>
<td>%a</td>
<td>Mon Sep 22 12:19:47 EDT 1986</td>
</tr>
<tr>
<td>Sun</td>
<td>%a</td>
<td>Sun Sep 28 12:19:47 EDT 1986</td>
</tr>
<tr>
<td>Fri</td>
<td>%a</td>
<td>Fri Sep 26 12:19:47 EDT 1986</td>
</tr>
<tr>
<td>September</td>
<td>%B</td>
<td>Mon Sep 1 12:19:47 EDT 1986</td>
</tr>
<tr>
<td>January</td>
<td>%B</td>
<td>Thu Jan 1 12:19:47 EST 1987</td>
</tr>
<tr>
<td>December</td>
<td>%B</td>
<td>Mon Dec 1 12:19:47 EST 1986</td>
</tr>
<tr>
<td>Sep Mon</td>
<td>%b %a</td>
<td>Mon Sep 1 12:19:47 EDT 1986</td>
</tr>
<tr>
<td>Jan Fri</td>
<td>%b %a</td>
<td>Fri Jan 2 12:19:47 EST 1987</td>
</tr>
<tr>
<td>Dec Mon</td>
<td>%b %a</td>
<td>Mon Dec 1 12:19:47 EST 1986</td>
</tr>
<tr>
<td>Jan Wed 1989</td>
<td>%b %a %Y</td>
<td>Wed Jan 4 12:19:47 EST 1989</td>
</tr>
<tr>
<td>Fri 9</td>
<td>%a %H</td>
<td>Fri Sep 26 09:00:00 EDT 1986</td>
</tr>
<tr>
<td>Feb 10:30</td>
<td>%b %H:%S</td>
<td>Sun Feb 1 10:00:30 EST 1987</td>
</tr>
<tr>
<td>10:30</td>
<td>%H:%M</td>
<td>Tue Sep 23 10:30:00 EDT 1986</td>
</tr>
<tr>
<td>13:30</td>
<td>%H:%M</td>
<td>Mon Sep 22 13:30:00 EDT 1986</td>
</tr>
</tbody>
</table>

APPLICATION USAGE
Although historical versions of getdate() did not require that <time.h> declare the external variable getdate_err, this document does require it. X/Open encourages applications to remove declarations of getdate_err and instead incorporate the declaration by including <time.h>.

SEE ALSO  
ctime(), strftime(), localtime(), setlocale(), strftime(), times, <time.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
getdtablesize( )

NAME
getdtablesize — get the file descriptor table size

SYNOPSIS

UX
#include <unistd.h>

int getdtablesize(void);

DESCRIPTION

The getdtablesize() function is equivalent to getrlimit() with the RLIMIT_NOFILE option.

RETURN VALUE

The getdtablesize() function returns the current soft limit as if obtained from a call to getrlimit() with the RLIMIT_NOFILE option.

ERRORS

No errors are defined.

APPLICATION USAGE

There is no direct relationship between the value returned by getdtablesize() and {OPEN_MAX} defined in <limits.h>.

SEE ALSO

close(), getrlimit(), open(), select(), setrlimit(), <limits.h>, <unistd.h>.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME

getegid — get effective group ID

SYNOPSIS

OH

#include <sys/types.h>
#include <unistd.h>

gid_t getegid(void);

DESCRIPTION

The getegid() function returns the effective group ID of the calling process.

RETURN VALUE

The getegid() function is always successful and no return value is reserved to indicate an error.

ERRORS

No errors are defined.

SEE ALSO

gid(), setgid(), <sys/types.h>, <unistd.h>.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following change is incorporated for alignment with the ISO POSIX-1 standard:

• The argument list is explicitly defined as void.

Other changes are incorporated as follows:

• The header <sys/types.h> is now marked as optional (OH); this header need not be included on XSI-conformant systems.

• The header <unistd.h> is added to the SYNOPSIS section.
NAME
getenv — get value of environment variable

SYNOPSIS
#include <stdlib.h>
char *getenv(const char *name);

DESCRIPTION
The getenv() function searches the environment list for a string of the form "name=value", and returns a pointer to a string containing the value for the specified name. If the specified name cannot be found, a null pointer is returned. The string pointed to must not be modified by the application, but may be overwritten by a subsequent call to getenv() or putenv() but will not be overwritten by a call to any other function in this document.

RETURN VALUE
Upon successful completion, getenv() returns a pointer to a string containing the value for the specified name. If the specified name cannot be found a null pointer is returned.

ERRORS
No errors are defined.

APPLICATION USAGE
The return value from getenv() may point to static data which may be overwritten by subsequent calls to getenv() or putenv().

SEE ALSO
dep, putenv(), <stdlib.h>, the XBD specification, Chapter 6, Environment Variables.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:
• The type of argument name is changed from char * to const char *.

Other changes are incorporated as follows:
• The DESCRIPTION section is updated to indicate that the return string (a) must not be modified by an application, (b) may be overwritten by subsequent calls to getenv() or putenv(), and (c) will not be overwritten by calls to other XSI system interfaces. A reference to putenv() has also been added to the APPLICATION USAGE section.
NAME
geteuid — get effective user ID

SYNOPSIS
\#include <sys/types.h>
\#include <unistd.h>

uid_t geteuid(void);

DESCRIPTION
The geteuid() function returns the effective user ID of the calling process.

RETURN VALUE
The geteuid() function is always successful and no return value is reserved to indicate an error.

ERRORS
No errors are defined.

SEE ALSO
getuid(), setuid(), <sys/types.h>, <unistd.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:

- The argument list is explicitly defined as void.

Other changes are incorporated as follows:

- The header <sys/types.h> is now marked as optional (OH); this header need not be included on XSI-conformant systems.
- The header <unistd.h> is added to the SYNOPSIS section.
NAME
getgid — get real group ID

SYNOPSIS

#include <sys/types.h>
#include <unistd.h>
gid_t getgid(void);

DESCRIPTION
The getgid() function returns the real group ID of the calling process.

RETURN VALUE
The getgid() function is always successful and no return value is reserved to indicate an error.

ERRORS
No errors are defined.

SEE ALSO
getuid(), setgid(), <sys/types.h>, <unistd.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:
- The argument list is explicitly defined as void.

Other changes are incorporated as follows:
- The header <sys/types.h> is now marked as optional (OH); this header need not be included on XSI-conformant systems.
- The header <unistd.h> is added to the SYNOPSIS section.
NAME
    getgrent — get group database entry

SYNOPSIS
    #include <grp.h>

    struct group *getgrent(void);

DESCRIPTION
    Refer to endgrent().

CHANGE HISTORY
    First released in Issue 4, Version 2.
getgrgid() — get group database entry for particular group ID

SYNOPSIS

```c
#include <sys/types.h>
#include <grp.h>

struct group *getgrgid(gid_t gid);
```

DESCRIPTION

The `getgrgid()` function searches the group database for an entry with a matching `gid`.

RETURN VALUE

Upon successful completion, `getgrgid()` returns a pointer to a `struct group` with the structure defined in `<grp.h>` with a matching entry if one is found. The `getgrgid()` function returns a null pointer if either the requested entry was not found, or an error occurred. On error, `errno` will be set to indicate the error.

ERRORS

The `getgrgid()` function may fail if:

- `[EIO]` An I/O error has occurred.
- `[EINTR]` A signal was caught during `getgrgid()`.
- `[EMFILE]` [OPEN_MAX] file descriptors are currently open in the calling process.
- `[ENFILE]` The maximum allowable number of files is currently open in the system.

APPLICATION USAGE

The return value may point to a static area which is overwritten by a subsequent call to `getgrent()`, `getgrgid()` or `getgrnam()`.

Applications wishing to check for error situations should set `errno` to 0 before calling `getgrgid()`. If `errno` is set on return, an error occurred.

SEE ALSO

`endgrent()`, `getgrnam()`, `<grp.h>`, `<limits.h>`, `<sys/types.h>`.

CHANGE HISTORY

First released in Issue 1.
Derived from System V Release 2.0.

Issue 4

The following changes are incorporated in this issue:

- The DESCRIPTION section is clarified.
- In the RETURN VALUE section, the reference to the setting of `errno` is marked as an extension.
- The errors `[EIO]`, `[EINTR]`, `[EMFILE]` and `[ENFILE]` are marked as extensions.
- A note is added to the APPLICATION USAGE section advising how applications should check for errors.
- The header `<sys/types.h>` is added as optional (OH); this header need not be included on XSI-conformant systems.
NAME
getgrnam — search group database for particular name

SYNOPSIS
OH
#include <sys/types.h>
#include <grp.h>

struct group *getgrnam(const char *name);

DESCRIPTION
The getgrnam() function searches the group database for an entry with a matching name.

RETURN VALUE
The getgrnam() function returns a pointer to a struct group with the structure defined in <grp.h> with a matching entry if one is found. The getgrnam() function returns a null pointer if either the requested entry was not found, or an error occurred. On error, errno will be set to indicate the error.

ERRORS
The getgrnam() function may fail if:

- [EIO] An I/O error has occurred.
- [EINTR] A signal was caught during getgrnam().
- [EMFILE] [OPEN_MAX] file descriptors are currently open in the calling process.
- [ENFILE] The maximum allowable number of files is currently open in the system.

APPLICATION USAGE
The return value may point to a static area which is overwritten by a subsequent call to getgrent(), getgrgid() or getgrnam().

Applications wishing to check for error situations should set errno to 0 before calling getgrnam(). If errno is set on return, an error occurred.

SEE ALSO
endgrent(), getgrgid(), <grp.h>, <limits.h>, <sys/types.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from System V Release 2.0.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:
- The type of argument name is changed from char * to const char *.

Other changes are incorporated as follows:
- The DESCRIPTION section is clarified.
- The header <sys/types.h> is added as optional (OH); this header need not be included on XSI-conformant systems.
- In the RETURN VALUE section, reference to the setting of errno is marked as an extension.
- The errors [EIO], [EINTR], [EMFILE] and [ENFILE] are marked as extensions.
- A note is added to the APPLICATION USAGE section advising how applications should check for errors.
NAME
getgroups — get supplementary group IDs

SYNOPSIS
#include <sys/types.h>
#include <unistd.h>

int getgroups(int gidsetsize, gid_t grouplist[]);

DESCRIPTION
The getgroups() function fills in the array grouplist with the current supplementary group IDs of
the calling process.

The gidsetsize argument specifies the number of elements in the array grouplist. The actual
number of supplementary group IDs stored in the array is returned. The values of array entries
with indices greater than or equal to the value returned are undefined.

If gidsetsize is 0, getgroups() returns the number of supplementary group IDs associated with the
calling process without modifying the array pointed to by grouplist.

RETURN VALUE
Upon successful completion, the number of supplementary group IDs is returned. A return
value of −1 indicates failure and errno is set to indicate the error.

ERRORS
The getgroups() function will fail if:

[EINVAL] The gidsetsize argument is non-zero and is less than the number of
supplementary group IDs.

APPLICATION USAGE
It is unspecified whether the effective group ID of the calling process is included in, or omitted
from, the returned list of supplementary group IDs.

SEE ALSO
getegid(), setgid(), <sys/types.h>, <unistd.h>.

CHANGE HISTORY
First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

Issue 4
The following change is incorporated for alignment with the FIPS requirements:

• A return value of 0 is no longer permitted, because [NGROUPS_MAX] cannot be 0.

Other changes are incorporated as follows:

• The header <sys/types.h> is now marked as optional (OH); this header need not be included
  on XSI-conformant systems.

• The header <unistd.h> is added to the SYNOPSIS section.
NAME
gethostid — get an identifier for the current host

SYNOPSIS

```c
#include <unistd.h>

long gethostid(void);
```

DESCRIPTION

The `gethostid()` function retrieves a 32-bit identifier for the current host.

RETURN VALUE

Upon successful completion, `gethostid()` returns an identifier for the current host.

ERRORS

No errors are defined.

APPLICATION USAGE

X/Open does not define the domain in which the return value is unique.

SEE ALSO

`random()`, `<unistd.h>`.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
getitimer, setitimer — get/set value of interval timer

SYNOPSIS
UX
#include <sys/time.h>

int getitimer(int which, struct itimerval *value);
int setitimer(int which, const struct itimerval *value,
              struct itimerval *ovalue);

DESCRIPTION
The getitimer() function stores the current value of the timer specified by which into the structure pointed to by value. The setitimer() function sets the timer specified by which to the value specified in the structure pointed to by value, and if ovalue is not a null pointer, stores the previous value of the timer in the structure pointed to by ovalue.

A timer value is defined by the itimerval structure. If it_value is non-zero, it indicates the time to the next timer expiration. If it_interval is non-zero, it specifies a value to be used in reloading it_value when the timer expires. Setting it_value to 0 disables a timer, regardless of the value of it_interval. Setting it_interval to 0 disables a timer after its next expiration (assuming it_value is non-zero).

Implementations may place limitations on the granularity of timer values. For each interval timer, if the requested timer value requires a finer granularity than the implementation supports, the actual timer value will be rounded up to the next supported value.

An XSI-conforming implementation provides each process with at least three interval timers, which are indicated by the which argument:

ITIMER_REAL
Decrement in real time. A SIGALRM signal is delivered when this timer expires.

ITIMER_VIRTUAL
Decrement in process virtual time. It runs only when the process is executing. A SIGVTALRM signal is delivered when it expires.

ITIMER_PROF
Decrement both in process virtual time and when the system is running on behalf of the process. It is designed to be used by interpreters in statistically profiling the execution of interpreted programs. Each time the ITIMER_PROF timer expires, the SIGPROF signal is delivered.

The interaction between setitimer() and any of alarm(), sleep() or usleep() is unspecified.

RETURN VALUE
Upon successful completion, getitimer() or setitimer() returns 0. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS
The setitimer() function will fail if:

[EINVAL] The value argument is not in canonical form. (In canonical form, the number of microseconds is a non-negative integer less than 1,000,000 and the number of seconds is a non-negative integer.)

The getitimer() and setitimer() functions may fail if:

[EINVAL] The which argument is not recognised.
SEE ALSO

alarm(), sleep(), ualarm(), usleep(), <signal.h>, <sys/time.h>.

CHANGE HISTORY

First released in Issue 4, Version 2.
getlogin()

NAME
getlogin — get login name

SYNOPSIS
#include <unistd.h>
char *getlogin(void);

DESCRIPTION
The getlogin() function returns a pointer to a string giving a user name associated with the calling process, which is the login name associated with the calling process. If getlogin() returns a non-null pointer, then that pointer points to the name that the user logged in under, even if there are several login names with the same user ID.

RETURN VALUE
Upon successful completion, getlogin() returns a pointer to the login name or a null pointer if the user's login name cannot be found. Otherwise it returns a null pointer and sets errno to indicate the error.

ERRORS
The getlogin() function may fail if:

- [EMFILE] [OPEN_MAX] file descriptors are currently open in the calling process.
- [ENFILE] The maximum allowable number of files is currently open in the system.
- [ENXIO] The calling process has no controlling terminal.

APPLICATION USAGE
The return value may point to static data whose content is overwritten by each call.

Three names associated with the current process can be determined: getpwuid(geteuid()) returns the name associated with the effective user ID of the process; getlogin() returns the name associated with the current login activity; and getpwuid(getuid()) returns the name associated with the real user ID of the process.

SEE ALSO
getpwnam(), getpwuid(), geteuid(), getuid(), <limits.h>, <unistd.h>.

CHANGE HISTORY
First released in Issue 1.

Derived from System V Release 2.0.

Issue 4
The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- The argument list is explicitly defined as void.
- The DESCRIPTION section is updated to state explicitly that the return value is a pointer to a string giving the user name, rather than simply a pointer to the user name as stated in previous issues.

Other changes are incorporated as follows:

- The header <unistd.h> is added to the SYNOPSIS section.
- In the RETURN VALUE section, reference to the setting of errno is marked as an extension.
- The behaviour of the function when the login name cannot be found is included in the RETURN VALUE section instead of the DESCRIPTION section.
• The errors [EMFILE], [ENFILE] and [ENXIO] are marked as extensions.

• The **APPLICATION USAGE** section is changed to refer to `getpwuid()` rather than `cuserid()`, which will be withdrawn in a future issue.
NAME
getmsg, getpmsg — receive next message from a STREAMS file

SYNOPSIS

```c
#include <stropts.h>

int getmsg(int fildes, struct strbuf *ctlptr, struct strbuf *dataptr,
            int *flagsp);

int getpmsg(int fildes, struct strbuf *ctlptr, struct strbuf *dataptr,
            int *bandp, int *flagsp);
```

DESCRIPTION

The `getmsg()` function retrieves the contents of a message located at the head of the STREAM head read queue associated with a STREAMS file and places the contents into one or more buffers. The message contains either a data part, a control part, or both. The data and control parts of the message are placed into separate buffers, as described below. The semantics of each part is defined by the originator of the message.

The `getpmsg()` function does the same thing as `getmsg()`, but provides finer control over the priority of the messages received. Except where noted, all requirements on `getmsg()` also pertain to `getpmsg()`.

The `fildes` argument specifies a file descriptor referencing a STREAMS-based file.

The `ctlptr` and `dataptr` arguments each point to a `strbuf` structure, in which the `buf` member points to a buffer in which the data or control information is to be placed, and the `maxlen` member indicates the maximum number of bytes this buffer can hold. On return, the `len` member contains the number of bytes of data or control information actually received. The `len` member is set to 0 if there is a zero-length control or data part and `len` is set to −1 if no data or control information is present in the message.

When `getmsg()` is called, `flagsp` should point to an integer that indicates the type of message the process is able to receive. This is described further below.

The `ctlptr` argument is used to hold the control part of the message, and `dataptr` is used to hold the data part of the message. If `ctlptr` (or `dataptr`) is a null pointer or the `maxlen` member is −1, the control (or data) part of the message is not processed and is left on the STREAM head read queue, and if the `ctlptr` (or `dataptr`) is not a null pointer, `len` is set to −1. If the `maxlen` member is set to 0 and there is a zero-length control or data part and `len` is set to −1 if no data or control information is present in the message.

When `getmsg()` is called, `flagsp` should point to an integer that indicates the type of message the process is able to receive. This is described further below.

By default, `getmsg()` processes the first available message on the STREAM head read queue. However, a process may choose to retrieve only high-priority messages by setting the integer pointed to by `flagsp` to RS_HIPRI. In this case, `getmsg()` will only process the next message if it is a high-priority message. When the integer pointed to by `flagsp` is 0, any message will be retrieved. In this case, on return, the integer pointed to by `flagsp` will be set to RS_HIPRI if a high-priority message was retrieved, or 0 otherwise.

For `getpmsg()`, the flags are different. The `flagsp` argument points to a bitmask with the following mutually-exclusive flags defined: MSG_HIPRI, MSG_BAND, and MSG_ANY. Like `getmsg()`, `getpmsg()` processes the first available message on the STREAM head read queue. A process may choose to retrieve only high-priority messages by setting the integer pointed to by `flagsp` to MSG_HIPRI and the integer pointed to by `bandp` to 0. In this case, `getpmsg()` will only process
the next message if it is a high-priority message. In a similar manner, a process may choose to retrieve a message from a particular priority band by setting the integer pointed to by flagsp to MSG_BAND and the integer pointed to by bandp to the priority band of interest. In this case, getpmsg() will only process the next message if it is in a priority band equal to, or greater than, the integer pointed to by bandp, or if it is a high-priority message. If a process just wants to get the first message off the queue, the integer pointed to by flagsp should be set to MSG_ANY and the integer pointed to by bandp should be set to 0. On return, if the message retrieved was a high-priority message, the integer pointed to by flagsp will be set to MSG_HIPRI and the integer pointed to by bandp will be set to 0. Otherwise, the integer pointed to by flagsp will be set to MSG_BAND and the integer pointed to by bandp will be set to the priority band of the message.

If O_NONBLOCK is not set, getmsg() and getpmsg() will block until a message of the type specified by flagsp is available at the front of the STREAM head read queue. If O_NONBLOCK is set and a message of the specified type is not present at the front of the read queue, getmsg() and getpmsg() fail and set errno to [EAGAIN].

If a hangup occurs on the STREAM from which messages are to be retrieved, getmsg() and getpmsg() continue to operate normally, as described above, until the STREAM head read queue is empty. Thereafter, they return 0 in the len members of ctlptr and dataptr.

RETURN VALUE

Upon successful completion, getmsg() and getpmsg() return a non-negative value. A value of 0 indicates that a full message was read successfully. A return value of MORECTL indicates that more control information is waiting for retrieval. A return value of MOREDATA indicates that more data is waiting for retrieval. A return value of the bitwise logical OR of MORECTL and MOREDATA indicates that both types of information remain. Subsequent getmsg() and getpmsg() calls retrieve the remainder of the message. However, if a message of higher priority has come in on the STREAM head read queue, the next call to getmsg() or getpmsg() retrieves that higher-priority message before retrieving the remainder of the previous message.

Upon failure, getmsg() and getpmsg() return −1 and set errno to indicate the error.

ERRORS

The getmsg() and getpmsg() functions will fail if:

[EAGAIN] The O_NONBLOCK flag is set and no messages are available.
[EBADF] The fildes argument is not a valid file descriptor open for reading.
[EBADMSG] The queued message to be read is not valid for getmsg() or getpmsg() or a pending file descriptor is at the STREAM head.
[EINTR] A signal was caught during getmsg() or getpmsg().
[EINVAL] An illegal value was specified by flagsp, or the STREAM or multiplexer referenced by fildes is linked (directly or indirectly) downstream from a multiplexer.
[ENOSTR] A STREAM is not associated with fildes.

In addition, getmsg() and getpmsg() will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of errno does not reflect the result of getmsg() or getpmsg() but reflects the prior error.

SEE ALSO

poll(), putmsg(), read(), write(), <stropts.h>, Section 2.5 on page 35.
CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
getopt, optarg, optind, opterr, optopt — command option parsing

SYNOPSIS
#include <unistd.h>

int getopt(int argc, char * const argv[], const char * optstring);
extern char *optarg;
extern int optind, opterr, optopt;

DESCRIPTION
The getopt() function is a command-line parser that can be used by applications that follow Utility Syntax Guidelines 3, 4, 5, 6, 7, 9 and 10 in the XBD specification, Section 10.2, Utility Syntax Guidelines. The remaining guidelines are not addressed by getopt() and are the responsibility of the application.

The parameters argc and argv are the argument count and argument array as passed to main() (see exec). The argument optstring is a string of recognised option characters; if a character is followed by a colon, the option takes an argument. All option characters allowed by Utility Syntax Guideline 3 are allowed in optstring. The implementation may accept other characters as an extension.

The variable optind is the index of the next element of the argv[] vector to be processed. It is initialised to 1 by the system, and getopt() updates it when it finishes with each element of argv[]. When an element of argv[] contains multiple option characters, it is unspecified how getopt() determines which options have already been processed.

The getopt() function returns the next option character (if one is found) from argv that matches a character in optstring, if there is one that matches. If the option takes an argument, getopt() sets the variable optarg to point to the option-argument as follows:

1. If the option was the last character in the string pointed to by an element of argv, then optarg contains the next element of argv, and optind is incremented by 2. If the resulting value of optind is not less than argc, this indicates a missing option-argument, and getopt() returns an error indication.

2. Otherwise, optarg points to the string following the option character in that element of argv, and optind is incremented by 1.

If, when getopt() is called:

argv[optind] is a null pointer
*argv[optind] is not the character –
argv[optind] points to the string "−"

g getopt() returns −1 without changing optind. If:

argv[optind] points to the string "−−"

g getopt() returns −1 after incrementing optind.

If getopt() encounters an option character that is not contained in optstring, it returns the question-mark (?) character. If it detects a missing option-argument, it returns the colon character (:) if the first character of optstring was a colon, or a question-mark character (?) otherwise. In either case, getopt() will set the variable optopt to the option character that caused the error. If the application has not set the variable opterr to 0 and the first character of optstring is not a colon, getopt() also prints a diagnostic message to stderr in the format specified for the getopts utility.
RETURN VALUE
The getopt() function returns the next option character specified on the command line.

A colon (:) is returned if getopt() detects a missing argument and the first character of optstring was a colon (:).

A question mark (?) is returned if getopt() encounters an option character not in optstring or detects a missing argument and the first character of optstring was not a colon (:).

Otherwise getopt() returns −1 when all command line options are parsed.

ERRORS
No errors are defined.

EXAMPLES
The following code fragment shows how one might process the arguments for a utility that can take the mutually exclusive options a and b and the options f and o, both of which require arguments:

```c
#include <unistd.h>

int main (int argc, char *argv[ ])
{
    int c;
    int bflg, aflg, errflg;
    char *ifile;
    char *ofile;
    extern char *optarg;
    extern int optind, optopt;
    ...
    while ((c = getopt(argc, argv, "abf:o:")) != -1) {
        switch (c) {
            case 'a':
                if (bflg)
                    errflg++;
                else
                    aflg++;
                break;
            case 'b':
                if (aflg)
                    errflg++;
                else {
                    bflg++;
                    bproc();
                }
                break;
            case 'f':
                ifile = optarg;
                break;
            case 'o':
                ofile = optarg;
                break;
            case ':': /* -f or -o without operand */
                fprintf(stderr,
                    "Option -%c requires an operand\n", optopt);
                errflg++;
                break;
            case '?':
                fprintf(stderr,
```
"Unrecognised option: -%c\n", optopt);
errflg++;
}
}
if (errflg) {
fprintf(stderr, "usage: . . . ");
exit(2);
}
for ( ; optind < argc; optind++) {
if (access(argv[optind], R_OK)) {...
...}

This code accepts any of the following as equivalent:

cmd -ao arg path path
cmd -a -o arg path path
cmd -o arg -a path path
cmd -a -o arg -- path path
cmd -a -o arg path path

APPLICATION USAGE
The getopt() function is only required to support option characters included in Guideline 3. Many historical implementations of getopt() support other characters as options. This is an allowed extension, but applications that use extensions are not maximally portable. Note that support for multi-byte option characters is only possible when such characters can be represented as type int.

SEE ALSO
exec, getopts, <unistd.h>, the XCU specification.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated for alignment with the ISO POSIX-2 standard:
• The header <unistd.h> is added to the SYNOPSIS section and <stdio.h> is deleted.
• The type of argument argv is changed from char ** to char * const [].
• The integer optopt is added to the list of external data items.
• The DESCRIPTION section is largely rewritten, without functional change, for alignment with the ISO POSIX-2 standard, although the following differences should be noted:
  — If the function detects a missing option-argument, it returns a colon (:) and sets optopt to the option character.
  — The termination conditions under which getopt() will return −1 are extended. Also note that the termination condition is explicitly −1, rather than the value of EOF.
• The EXAMPLES section is changed to illustrate the new functionality.
getpagesize()  

NAME
getpagesize — get the current page size

SYNOPSIS
UX

```c
#include <unistd.h>

int getpagesize(void);
```

DESCRIPTION
The `getpagesize()` function returns the current page size.

The `getpagesize()` function is equivalent to `sysconf(_SC_PAGE_SIZE)` and `sysconf(_SC_PAGESIZE)`.

RETURN VALUE
The `getpagesize()` function returns the current page size.

ERRORS
No errors are defined.

APPLICATION USAGE
The value returned by `getpagesize()` need not be the minimum value that `malloc()` can allocate. Moreover, the application cannot assume that an object of this size can be allocated with `malloc()`.

SEE ALSO
`brk()`, `getrlimit()`, `mmap()`, `mprotect()`, `munmap()`, `msync()`, `sysconf()`, `<unistd.h>`.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
getpass — read a string of characters without echo (TO BE WITHDRAWN)

SYNOPSIS

```c
#include <unistd.h>

char *getpass(const char *prompt);
```

DESCRIPTION
The `getpass()` function opens the process' controlling terminal, writes to that device the null-terminated string `prompt`, disables echoing, reads a string of characters up to the next newline character or EOF, restores the terminal state and closes the terminal.

RETURN VALUE
Upon successful completion, `getpass()` returns a pointer to a null-terminated string of at most `PASS_MAX` bytes that were read from the terminal device. If an error is encountered, the terminal state is restored and a null pointer is returned.

ERRORS
The `getpass()` function may fail if:

```
[EINTR] The `getpass()` function was interrupted by a signal.
[EIO] The process is a member of a background process attempting to read from its controlling terminal, the process is ignoring or blocking the SIGTTOU signal or the process group is orphaned. This error may also be generated for implementation-dependent reasons.
[EMFILE] `OPEN_MAX` file descriptors are currently open in the calling process.
[ENFILE] The maximum allowable number of files is currently open in the system.
[ENXIO] The process does not have a controlling terminal.
```

APPLICATION USAGE
The return value points to static data whose content may be overwritten by each call.

This interface is marked TO BE WITHDRAWN because its name is misleading, and it provides no functionality which the user could not easily implement.

SEE ALSO
`<limits.h>`, `<unistd.h>`.

CHANGE HISTORY
First released in Issue 1.

Derived from System V Release 2.0.

Issue 4
The following changes are incorporated in this issue:

- The interface is marked TO BE WITHDRAWN, because of its misleading name and because it provides dubious functionality.
- The header `<unistd.h>` is added to the SYNOPSIS section.
- The type of argument `prompt` is changed from `char *` to `const char *`.
- In the DESCRIPTION section, reference to the character special file `/dev/tty` is replaced by the phrase “the process’ controlling terminal”.
• In the **RETURN VALUE** section, the word “characters” is replaced by “bytes”, to indicate that this interface deals solely in single-byte values.

• A note is added to the **APPLICATION USAGE** section indicating why the interface is to be withdrawn.
NAME
getpgid — get process group ID

SYNOPSIS

```c
#include <unistd.h>

pid_t getpgid(pid_t pid);
```

DESCRIPTION
The `getpgid()` function returns the process group ID of the process whose process ID is equal to `pid`. If `pid` is equal to 0, `getpgid()` returns the process group ID of the calling process.

RETURN VALUE
Upon successful completion, `getpgid()` returns a process group ID. Otherwise, it returns `(pid_t)-1` and sets `errno` to indicate the error.

ERRORS
The `getpgid()` function will fail if:

- `[EPERM]` The process whose process ID is equal to `pid` is not in the same session as the calling process, and the implementation does not allow access to the process group ID of that process from the calling process.
- `[ESRCH]` There is no process with a process ID equal to `pid`.

The `getpgid()` function may fail if:

- `[EINVAL]` The value of the `pid` argument is invalid.

SEE ALSO
`exec`, `fork()`, `getpgid()`, `getpgrp()`, `getsid()`, `setpgid()`, `setsid()`, `<unistd.h>`.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
getpgrp — get process group ID

SYNOPSIS
#include <sys/types.h>
#include <unistd.h>

pid_t getpgrp(void);

DESCRIPTION
The getpgrp() function returns the process group ID of the calling process.

RETURN VALUE
The getpgrp() function is always successful and no return value is reserved to indicate an error.

ERRORS
No errors are defined.

SEE ALSO
exec, fork(), getpgid(), getpid(), getppid(), kill(), setpgid(), setsid(), <sys/types.h>, <unistd.h>.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:

• The argument list is explicitly defined as void.

Other changes are incorporated in this issue as follows:

• The header <sys/types.h> is now marked as optional (OH); this header need not be included on XSI-conformant systems.

• The header <unistd.h> is added to the SYNOPSIS section.
NAME
getpid — get process ID

SYNOPSIS
OH
#include <sys/types.h>
#include <unistd.h>

pid_t getpid(void);

DESCRIPTION
The getpid() function returns the process ID of the calling process.

RETURN VALUE
The getpid() function is always successful and no return value is reserved to indicate an error.

ERRORS
No errors are defined.

SEE ALSO
exec, fork(), getpgid(), getppid(), kill(), setpgid(), setsid(), <sys/types.h>, <unistd.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:
• The argument list is explicitly defined as void.
Other changes are incorporated in this issue as follows:
• The header <sys/types.h> is now marked as optional (OH); this header need not be included on XSI-conformant systems.
• The header <unistd.h> is added to the SYNOPSIS section.
getpmsg() — get user database entry

SYNOPSIS

UX

```c
#include <pwd.h>

int getpmsg(int fildes, struct strbuf *ctlptr, struct strbuf *dataptr,
            int *bandp, int *flagsp);
```

DESCRIPTION

Refer to `getmsg()`.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
  getppid — get parent process ID

SYNOPSIS
  #include <sys/types.h>
  #include <unistd.h>
  pid_t getppid(void);

DESCRIPTION
  The getppid() function returns the parent process ID of the calling process.

RETURN VALUE
  The getppid() function is always successful and no return value is reserved to indicate an error.

ERRORS
  No errors are defined.

SEE ALSO
  exec, fork(), getpgid(), getpgrp(), getpid(), kill(), setpgid(), setsid(), <sys/types.h>, <unistd.h>.

CHANGE HISTORY
  First released in Issue 1.
  Derived from Issue 1 of the SVID.

Issue 4
  The following change is incorporated for alignment with the ISO POSIX-1 standard:
    • The argument list is explicitly defined as void.
  Other changes are incorporated in this issue as follows:
    • The header <sys/types.h> is now marked as optional (OH); this header need not be included on XSI-conformant systems.
    • The header <unistd.h> is added to the SYNOPSIS section.
NAME
getpriority, setpriority — get or set process scheduling priority

SYNOPSIS

```c
#include <sys/resource.h>

int getpriority(int which, id_t who);

int setpriority(int which, id_t who, int priority);
```

DESCRIPTION

The `getpriority()` function obtains the current scheduling priority of a process, process group, or user. The `setpriority()` function sets the scheduling priority of a process, process group, or user.

Target processes are specified by the values of the `which` and `who` arguments. The `which` argument may be one of the following values: PRIO_PROCESS, PRIO_PGRP, or PRIO_USER, indicating that the `who` argument is to be interpreted as a process ID, a process group ID, or a user ID, respectively. A 0 value for the `who` argument specifies the current process, process group, or user.

If more than one process is specified, `getpriority()` returns the highest priority (lowest numerical value) pertaining to any of the specified processes, and `setpriority()` sets the priorities of all of the specified processes to the specified value.

The default priority is 0; negative priorities cause more favourable scheduling. While the range of valid priority values is `[-20, 20]`, implementations may enforce more restrictive limits. If the value specified to `setpriority()` is less than the system’s lowest supported priority value, the system’s lowest supported value is used; if it is greater than the system’s highest supported value, the system’s highest supported value is used.

Only a process with appropriate privileges can raise its priority (ie. assign a lower numerical priority value).

RETURN VALUE

Upon successful completion, `getpriority()` returns an integer in the range from -20 to 20. Otherwise, -1 is returned and `errno` is set to indicate the error.

Upon successful completion, `setpriority()` returns 0. Otherwise, -1 is returned and `errno` is set to indicate the error.

ERRORS

The `getpriority()` and `setpriority()` functions will fail if:

- **ESRCH** No process could be located using the `which` and `who` argument values specified.
- **EINVAL** The value of the `which` argument was not recognised, or the value of the `who` argument is not a valid process ID, process group ID, or user ID.

In addition, `setpriority()` may fail if:

- **EPERM** A process was located, but neither the real nor effective user ID of the executing process match the effective user ID of the process whose priority is being changed.
- **EACCES** A request was made to change the priority to a lower numeric value (that is, to a higher priority) and the current process does not have appropriate privileges.
APPLICATION USAGE

The effect of changing the scheduling priority may vary depending on the process-scheduling algorithm in effect.

Because `getpriority()` can return the value −1 on successful completion, it is necessary to set `errno` to 0 prior to a call to `getpriority()`. If `getpriority()` returns the value −1, then `errno` can be checked to see if an error occurred or if the value is a legitimate priority.

SEE ALSO

`nice()`, `<sys/resource.h>`.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
getpwent — get user database entry

SYNOPSIS

```c
#include <pwd.h>

struct passwd *getpwent(void);
```

DESCRIPTION
Refer to `endpwent()`.

CHANGE HISTORY
First released in Issue 4, Version 2.
getpwnam( )

NAME
getpwnam — search user database for particular name

SYNOPSIS
#include <sys/types.h>
#include <pwd.h>

struct passwd *getpwnam(const char *name);

DESCRIPTION
The getpwnam() function searches the user database for an entry with a matching name.

RETURN VALUE
The getpwnam() function returns a pointer to a struct passwd with the structure as defined in <pwd.h> with a matching entry if found. A null pointer is returned if the requested entry is not found, or an error occurs. On error, errno is set to indicate the error.

ERRORS
The getpwnam() function may fail if:

EX
[EINVAL] An I/O error has occurred.
[EINVAL] A signal was caught during getpwnam().
[ENOMEM] |OPEN_MAX| file descriptors are currently open in the calling process.
[ENOMEM] The maximum allowable number of files is currently open in the system.

APPLICATION USAGE
The return value may point to a static area which is overwritten by a subsequent call to cuserid(), getpwent(), getpwnam() or getpwuid().

Applications wishing to check for error situations should set errno to 0 before calling getpwnam(). If errno is set to non-zero on return, an error occurred.

Three names associated with the current process can be determined: getpwuid(geteuid()) returns the name associated with the effective user ID of the process; getlogin() returns the name associated with the current login activity; and getpwuid(getuid()) returns the name associated with the real user ID of the process.

SEE ALSO
getpwuid(), <limits.h>, <pwd.h>, <sys/types.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from System V Release 2.0.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:
- The type of argument name is changed from char * to const char *.
Other changes are incorporated as follows:
- The DESCRIPTION section is clarified.
- The header <sys/types.h> is now marked as optional (OH); this header need not be included on XSI-conformant systems.
- The last sentence in the RETURN VALUE section, indicating that errno will be set on error, is marked as an extension.
- The errors [EIO], [EINVAL], [EMFILE] and [ENFILE] are marked as extensions.

- The **APPLICATION USAGE** section is expanded (a) to warn about possible reuses of the area used to pass the return value, and (b) to indicate how applications should check for errors.
**NAME**
getpwuid — search user database for particular user ID

**SYNOPSIS**
```c
#include <sys/types.h>
#include <pwd.h>

struct passwd *getpwuid(uid_t uid);
```

**DESCRIPTION**
The `getpwuid()` function searches the user database for an entry with a matching `uid`.

**RETURN VALUE**
The `getpwuid()` function returns a pointer to a `struct passwd` with the structure as defined in `<pwd.h>` with a matching entry if found. A null pointer is returned if the requested entry is not found, or an error occurs. On error, `errno` is set to indicate the error.

**ERRORS**
The `getpwuid()` function may fail if:

- `[EIO]` An I/O error has occurred.
- `[EINTR]` A signal was caught during `getpwuid()`.
- `[EMFILE]` `[OPEN_MAX]` file descriptors are currently open in the calling process.
- `[ENFILE]` The maximum allowable number of files is currently open in the system.

**APPLICATION USAGE**
The return value may point to a static area which is overwritten by a subsequent call to `cuserid()`, `getpwnam()`, `getpwent()` or `getpwuid()`.

Applications wishing to check for error situations should set `errno` to 0 before calling `getpwuid()`. If `errno` is set to non-zero on return, an error occurred.

Three names associated with the current process can be determined: `getpwuid(geteuid())` returns the name associated with the effective user ID of the process; `getlogin()` returns the name associated with the current login activity; and `getpwuid(getuid())` returns the name associated with the real user ID of the process.

**SEE ALSO**
`cuserid()`, `getpwnam()`, `getuid()`, `geteuid()`, `getlogin()`, `<limits.h>`, `<pwd.h>`, `<sys/types.h>`.

**CHANGE HISTORY**
First released in Issue 1.

Derived from System V Release 2.0.

**Issue 4**
The following changes are incorporated in this issue:

- The `DESCRIPTION` section is clarified.
- The header `<sys/types.h>` is now marked as optional (OH); this header need not be included on XSI-conformant systems.
- The last sentence in the `RETURN VALUE` section, indicating that `errno` will be set on error, is marked as an extension.
• The errors [EIO], [EINTR], [EMFILE] and [ENFILE] are marked as extensions.

• A note is added to the **APPLICATION USAGE** section indicating how an application should check for errors.
NAME

getrlnit, setrlimit — control maximum resource consumption

SYNOPSIS

```c
#include <sys/resource.h>

int getrlimit(int resource, struct rlimit * rlp);
int setrlimit(int resource, const struct rlimit * rlp);
```

DESCRIPTION

Limits on the consumption of a variety of resources by the calling process may be obtained with `g`etrlinit() and set with `s`etrlimit().

Each call to either `getrlimit()` or `setrlimit()` identifies a specific resource to be operated upon as well as a resource limit. A resource limit is represented by an `rlimit` structure. The `rlim_cur` member specifies the current or soft limit and the `rlim_max` member specifies the maximum or hard limit. Soft limits may be changed by a process to any value that is less than or equal to the hard limit. A process may (irreversibly) lower its hard limit to any value that is greater than or equal to the soft limit. Only a process with appropriate privileges can raise a hard limit. Both hard and soft limits can be changed in a single call to `setrlimit()` subject to the constraints described above.

The value RLIM_INFINITY, defined in `<sys/resource.h>`, is considered to be larger than any other limit value. If a call to `getrlimit()` returns RLIM_INFINITY for a resource, it means the implementation does not enforce limits on that resource. Specifying RLIM_INFINITY as any resource limit value on a successful call to `setrlimit()` inhibits enforcement of that resource limit.

The following resources are defined:

- **RLIMIT_CORE**: This is the maximum size of a core file in bytes that may be created by a process. A limit of 0 will prevent the creation of a core file. If this limit is exceeded, the writing of a core file will terminate at this size.

- **RLIMIT_CPU**: This is the maximum amount of CPU time in seconds used by a process. If this limit is exceeded, SIGXCPU is generated for the process. If the process is blocking, catching or ignoring SIGXCPU, the behaviour is unspecified.

- **RLIMIT_DATA**: This is the maximum size of a process’ data segment in bytes. If this limit is exceeded, the `brk()`, `malloc()` and `sbrk()` functions will fail with `errno` set to [ENOMEM].

- **RLIMITFSIZE**: This is the maximum size of a file in bytes that may be created by a process. A limit of 0 will prevent the creation of a file. If a write or truncate operation would cause this limit to be exceeded, SIGXFSZ is generated for the process. If the process is blocking, catching or ignoring SIGXFSZ, continued attempts to increase the size of a file from end-of-file to beyond the limit will fail with `errno` set to [E2BIG].

- **RLIMIT_NFILE**: This is a number one greater than the maximum value that the system may assign to a newly-created descriptor. If this limit is exceeded, functions that allocate new file descriptors may fail with `errno` set to [EMFILE]. This limit constrains the number of file descriptors that a process may allocate.

- **RLIMIT_STACK**: This is the maximum size of a process’ stack in bytes. The implementation will not automatically grow the stack beyond this limit. If this limit is exceeded, SIGSEGV is generated for the process. If the process is blocking or ignoring SIGSEGV, or is catching SIGSEGV and has not made arrangements to
use an alternate stack, the disposition of SIGSEGV will be set to SIG_DFL before it is generated.

**RLIMIT_AS**

This is the maximum size of a process' total available memory, in bytes. If this limit is exceeded, the `brk()`, `malloc()`, `mmap()` and `sbrk()` functions will fail with `errno` set to [ENOMEM]. In addition, the automatic stack growth will fail with the effects outlined above.

**RETURN VALUE**

Upon successful completion, `getrlimit()` and `setrlimit()` return 0. Otherwise, these functions return −1 and set `errno` to indicate the error.

**ERRORS**

The `getrlimit()` and `setrlimit()` functions will fail if:

- **[EINVAL]** An invalid resource was specified; or in a `setrlimit()` call, the new `rlim_cur` exceeds the new `rlim_max`.

- **[EPERM]** The limit specified to `setrlimit()` would have raised the maximum limit value, and the calling process does not have appropriate privileges.

The `setrlimit()` function may fail if:

- **[EINVAL]** The limit specified cannot be lowered because current usage is already higher than the limit.

**SEE ALSO**

`brk()`, `exec`, `fork()`, `getdtablesize()`, `malloc()`, `open()`, `sigaltstack()`, `sysconf()`, `ulimit()`, `<stropts.h>`, `<sys/resource.h>`.

**CHANGE HISTORY**

First released in Issue 4, Version 2.
NAME
getrusage — get information about resource utilisation

SYNOPSIS
#include <sys/resource.h>

int getrusage(int who, struct rusage *r_usage);

DESCRIPTION
The getrusage() function provides measures of the resources used by the current process or its terminated and waited-for child processes. If the value of the who argument is RUSAGE_SELF, information is returned about resources used by the current process. If the value of the who argument is RUSAGE_CHILDREN, information is returned about resources used by the terminated and waited-for children of the current process. If the child is never waited for (for instance, if the parent has SA_NOCLDWAIT set or sets SIGCHLD to SIG_IGN), the resource information for the child process is discarded and not included in the resource information provided by getrusage().

The r_usage argument is a pointer to an object of type struct rusage in which the returned information is stored.

RETURN VALUE
Upon successful completion, getrusage() returns 0. Otherwise, −1 is returned, and errno is set to indicate the error.

ERRORS
The getrusage() function will fail if:
[EINVAL] The value of the who argument is not valid.

SEE ALSO
exit(), sigaction(), time(), times(), wait(), <sys/resource.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
gets — get a string from stdin stream

SYNOPSIS
#include <stdio.h>
char *gets(char *s);

DESCRIPTION
The gets() function reads bytes from the standard input stream, stdin, into the array pointed to by s, until a newline is read or an end-of-file condition is encountered. Any newline is discarded and a null byte is placed immediately after the last byte read into the array.

The gets() function may mark the st_atime field of the file associated with stream for update. The st_atime field will be marked for update by the first successful execution of fgetc(), fgets(), fread(), getc(), getchar(), gets(), fscanf() or scanf() using stream that returns data not supplied by a prior call to ungetc().

RETURN VALUE
Upon successful completion, gets() returns s. If the stream is at end-of-file, the end-of-file indicator for the stream is set and gets() returns a null pointer. If a read error occurs, the error indicator for the stream is set, gets() returns a null pointer and sets errno to indicate the error.

ERRORS
Refer to fgetc().

APPLICATION USAGE
Reading a line that overflows the array pointed to by s causes undefined results. The use of fgets() is recommended.

SEE ALSO
feof(), ferror(), fgets(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:
• In the DESCRIPTION section, (a) the text is changed to make it clear that the function reads bytes rather than (possibly multi-byte) characters, and (b) the list of functions that may cause the st_atime field to be updated is revised.
NAME
getsid — get process group ID of session leader

SYNOPSIS
UX
#include <unistd.h>

pid_t getsid(pid_t pid);

DESCRIPTION
The getsid() function obtains the process group ID of the process that is the session leader of the
process specified by pid. If pid is (pid_t)0, it specifies the calling process.

RETURN VALUE
Upon successful completion, getsid() returns the process group ID of the session leader of the
specified process. Otherwise, it returns (pid_t)−1 and sets errno to indicate the error.

ERRORS
The getsid() function will fail if:

[EPERM] The process specified by pid is not in the same session as the calling process,
and the implementation does not allow access to the process group ID of the
session leader of that process from the calling process.

[ESRCH] There is no process with a process ID equal to pid.

SEE ALSO
exec, fork(), getpid(), getpgid(), setpgid(), setsid(), <unistd.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
getsubopt — parse suboption arguments from a string

SYNOPSIS
UX

#include <stdlib.h>

int getsubopt(char **optionp, char * const *tokens, char **valuep);

DESCRIPTION
The getsubopt() function parses suboption arguments in a flag argument that was initially parsed
by getopt(). These suboption arguments must be separated by commas and may consist of
either a single token, or a token-value pair separated by an equal sign. Because commas delimit
suboption arguments in the option string, they are not allowed to be part of the suboption
arguments or the value of a suboption argument. Similarly, because the equal sign separates a
token from its value, a token must not contain an equal sign.

The getsubopt() function takes the address of a pointer to the option argument string, a vector of
possible tokens, and the address of a value string pointer. If the option argument string at
*optionp contains only one suboption argument, getsubopt() updates *optionp to point to the null
at the end of the string. Otherwise, it isolates the suboption argument by replacing the comma
separator with a null, and updates *optionp to point to the start of the next suboption argument.
If the suboption argument has an associated value, getsubopt() updates *valuep to point to the
value's first character. Otherwise it sets *valuep to a null pointer.

The token vector is organised as a series of pointers to strings. The end of the token vector is
identified by a null pointer.

When getsubopt() returns, if *valuep is not a null pointer then the suboption argument processed
included a value. The calling program may use this information to determine if the presence or
lack of a value for this suboption is an error.

Additionally, when getsubopt() fails to match the suboption argument with the tokens in the
tokens array, the calling program should decide if this is an error, or if the unrecognised option
should be passed on to another program.

RETURN VALUE
The getsubopt() function returns the index of the matched token string, or −1 if no token strings
were matched.

ERRORS
No errors are defined.

SEE ALSO
getopt(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
gettimeofday — get the date and time

SYNOPSIS
UX

#include <sys/time.h>

int gettimeofday(struct timeval * tp, void * tzp);

DESCRIPTION
The gettimeofday() function obtains the current time, expressed as seconds and microseconds since 00:00 Coordinated Universal Time (UTC), January 1, 1970, and stores it in the timeval structure pointed to by tp. The resolution of the system clock is unspecified.

If tzp is not a null pointer, the behaviour is unspecified.

RETURN VALUE
The gettimeofday() function returns 0 and no value is reserved to indicate an error.

ERRORS
No errors are defined.

SEE ALSO
ctime(), ftime(), <sys/time.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
getuid — get real user ID

SYNOPSIS
#include <sys/types.h>
#include <unistd.h>

uid_t getuid (void);

DESCRIPTION
The getuid() function returns the real user ID of the calling process.

RETURN VALUE
The getuid() function is always successful and no return value is reserved to indicate the error.

ERRORS
No errors are defined.

SEE ALSO
geteuid(), getgid(), setuid(), <sys/types.h>, <unistd.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:
- The argument list is explicitly defined as void.

Other changes are incorporated as follows:
- The header <sys/types.h> is now marked as optional (OH); this header need not be included on XSI-conformant systems.
- The header <unistd.h> is added to the SYNOPSIS section.
NAME
getutxent, getutxiid, getutxline — get user accounting database entries

SYNOPSIS
UX
#include <utmpx.h>
struct utmpx *getutxent(void);
struct utmpx *getutxiid(const struct utmpx *id);
struct utmpx *getutxline(const struct utmpx *line);

DESCRIPTION
Refer to endutxent().

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
getw — get a word from a stream

SYNOPSIS
EX
#define <stdio.h>
int getw(FILE *stream);

DESCRIPTION
The getw() function reads the next word from the stream. The size of a word is the size of an int and may vary from machine to machine. The getw() function presumes no special alignment in the file.

The getw() function may mark the st_atime field of the file associated with stream for update. The st_atime field will be marked for update by the first successful execution of fgetc(), fgets(), fread(), getc(), getchar(), gets(), fscanf() or scanf() using stream that returns data not supplied by a prior call to ungetc().

RETURN VALUE
Upon successful completion, getw() returns the next word from the input stream pointed to by stream. If the stream is at end-of-file, the end-of-file indicator for the stream is set and getw() returns EOF. If a read error occurs, the error indicator for the stream is set, getw() returns EOF and sets errno to indicate the error.

ERRORS
Refer to fgetc().

APPLICATION USAGE
Because of possible differences in word length and byte ordering, files written using putw() are machine-dependent, and may not be read using getw() on a different processor.

Because the representation of EOF is a valid integer, applications wishing to check for errors should use ferror() and feof().

SEE ALSO
feof(), ferror(), getc(), putw(), <stdio.h>, <utmpx.h>.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

• In the DESCRIPTION section, the list of functions that may cause the st_atime field to be updated is revised.

• The APPLICATION USAGE section is amended because EOF is always a valid integer.
NAME
getwc — get wide character from a stream

SYNOPSIS

OH
#include <stdio.h>
WP
#include <wchar.h>

wint_t getwc(FILE *stream);

DESCRIPTION
The getwc() function is equivalent to fgetwc(), except that if it is implemented as a macro it may evaluate stream more than once, so the argument should never be an expression with side effects.

RETURN VALUE
Refer to fgetwc().

ERRORS
Refer to fgetwc().

APPLICATION USAGE
This interface is provided in order to align with some current implementations, and with possible future ISO standards.

Because it may be implemented as a macro, getwc() may treat incorrectly a stream argument with side effects. In particular, getwc(*f++) may not work as expected. Therefore, use of this function is not recommended; fgetwc() should be used instead.

SEE ALSO
fgetwc(), <stdio.h>, <wchar.h>.

CHANGE HISTORY
First released as a World-wide Portability Interface in Issue 4.
Derived from the MSE working draft.
NAME

getwchar — get wide character from stdin stream

SYNOPSIS

```c
#include <wchar.h>

wint_t getwchar(void);
```

DESCRIPTION

The `getwchar` function is equivalent to `getwc(stdin)`.

RETURN VALUE

Refer to `fgetwc()`.

ERRORS

Refer to `fgetwc()`.

APPLICATION USAGE

If the value returned by `getwchar()` is stored into a variable of type `wchar_t` and then compared against the `wint_t` macro `WEOF`, the comparison may never succeed.

SEE ALSO

`fgetwc()`, `getwc()`, `<wchar.h>`.

CHANGE HISTORY

First released as a World-wide Portability Interface in Issue 4.

Derived from the MSE working draft.
NAME
getwd — get the current working directory pathname

SYNOPSIS
UX
#include <unistd.h>

char *getwd(char *path_name);

DESCRIPTION
The getwd() function determines an absolute pathname of the current working directory of the
calling process, and copies that pathname into the array pointed to by the path_name argument.

If the length of the pathname of the current working directory is greater than ({PATH_MAX} + 1)
including the null byte, getwd() fails and returns a null pointer.

RETURN VALUE
Upon successful completion, a pointer to the string containing the absolute pathname of the
current working directory is returned. Otherwise, getwd() returns a null pointer and the
contents of the array pointed to by path_name are undefined.

ERRORS
No errors are defined.

APPLICATION USAGE
For portability to implementations conforming to earlier versions of this document, getcwd() is
preferred over this function.

SEE ALSO
getcwd(), <unistd.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
glob, globfree — generate pathnames matching a pattern

SYNOPSIS
#include <glob.h>

int glob(const char * pattern, int flags,
         int (*errfunc)(const char * epath, int eerrno), glob_t *pglob);
void globfree(glob_t *pglob);

DESCRIPTION
The glob() function is a pathname generator that implements the rules defined in the XCU specification, Section 2.13, Pattern Matching Notation, with optional support for rule 3 in the XCU specification, Section 2.13.3, Patterns Used for Filename Expansion.

The structure type glob_t is defined in the header <glob.h> and includes at least the following members:

<table>
<thead>
<tr>
<th>Member Type</th>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_t</td>
<td>gl_pathc</td>
<td>Count of paths matched by pattern.</td>
</tr>
<tr>
<td>char **</td>
<td>gl_pathv</td>
<td>Pointer to a list of matched pathnames.</td>
</tr>
<tr>
<td>size_t</td>
<td>gl_offs</td>
<td>Slots to reserve at the beginning of gl_pathv.</td>
</tr>
</tbody>
</table>

The argument pattern is a pointer to a pathname pattern to be expanded. The glob() function matches all accessible pathnames against this pattern and develops a list of all pathnames that match. In order to have access to a pathname, glob() requires search permission on every component of a path except the last, and read permission on each directory of any filename component of pattern that contains any of the following special characters:

* ? [ ]

The glob() function stores the number of matched pathnames into pglob->gl_pathc and a pointer to a list of pointers to pathnames into pglob->gl_pathv. The pathnames are in sort order as defined by the current setting of the LC_COLLATE category, see the XBD specification, Section 5.3.2, LC_COLLATE. The first pointer after the last pathname is a null pointer. If the pattern does not match any pathnames, the returned number of matched paths is set to 0, and the contents of pglob->gl_pathv are implementation-dependent.

It is the caller’s responsibility to create the structure pointed to by pglob. The glob() function allocates other space as needed, including the memory pointed to by gl_pathv. The globfree() function frees any space associated with pglob from a previous call to glob().

The flags argument is used to control the behaviour of glob(). The value of flags is a bitwise inclusive OR of zero or more of the following constants, which are defined in the header <glob.h>:

- GLOB_APPEND Append pathnames generated to the ones from a previous call to glob().
- GLOB_DOOFFS Make use of pglob->gl_offs. If this flag is set, pglob->gl_offs is used to specify how many null pointers to add to the beginning of pglob->gl_pathv. In other words, pglob->gl_pathv will point to pglob->gl_offs null pointers, followed by pglob->gl_pathc pathname pointers, followed by a null pointer.
- GLOB_ERR Causes glob() to return when it encounters a directory that it cannot open or read. Ordinarily, glob() continues to find matches.
GLOB_MARK: Each pathname that is a directory that matches pattern has a slash appended.

GLOB_NOCHECK: Support rule 3 in the XCU specification, Section 2.13.3, Patterns Used for Filename Expansion. If pattern does not match any pathname, then glob() returns a list consisting of only pattern, and the number of matched pathnames is 1.

GLOB_NOESCAPE: Disable backslash escaping.

GLOB_NOSORT: Ordinarily, glob() sorts the matching pathnames according to the current setting of the LC_COLLATE category, see the XBD specification, Section 5.3.2, LC_COLLATE. When this flag is used the order of pathnames returned is unspecified.

The GLOB_APPEND flag can be used to append a new set of pathnames to those found in a previous call to glob(). The following rules apply when two or more calls to glob() are made with the same value of pglob and without intervening calls to globfree():

1. The first such call must not set GLOB_APPEND. All subsequent calls must set it.
2. All the calls must set GLOB_DOOFFS, or all must not set it.
3. After the second call, pglob->gl_pathv points to a list containing the following:
   a. Zero or more null pointers, as specified by GLOB_DOOFFS and pglob->gl_offs.
   b. Pointers to the pathnames that were in the pglob->gl_pathv list before the call, in the same order as before.
   c. Pointers to the new pathnames generated by the second call, in the specified order.
4. The count returned in pglob->gl_pathc will be the total number of pathnames from the two calls.
5. The application can change any of the fields after a call to glob(). If it does, it must reset them to the original value before a subsequent call, using the same pglob value, to globfree() or glob() with the GLOB_APPEND flag.

If, during the search, a directory is encountered that cannot be opened or read and errfunc is not a null pointer, glob() calls (*errfunc()) with two arguments:

1. The epath argument is a pointer to the path that failed.
2. The errno argument is the value of errno from the failure, as set by opendir(), readdir() or stat(). (Other values may be used to report other errors not explicitly documented for those functions.)

The following constants are defined as error return values for glob():

GLOB_ABORTED: The scan was stopped because GLOB_ERR was set or (*errfunc()) returned non-zero.

GLOB_NOMATCH: The pattern does not match any existing pathname, and GLOB_NOCHECK was not set in flags.

GLOB_NOSPACE: An attempt to allocate memory failed.

If (*errfunc()) is called and returns non-zero, or if the GLOB_ERR flag is set in flags, glob() stops the scan and returns GLOB_ABORTED after setting gl_pathc and gl_pathv in pglob to reflect the paths already scanned. If GLOB_ERR is not set and either errfunc is a null pointer or (*errfunc()) returns 0, the error is ignored.
RETURN VALUE

On successful completion, `glob()` returns 0. The argument `pglob->gl_pathc` returns the number of matched pathnames and the argument `pglob->gl_pathv` contains a pointer to a null-terminated list of matched and sorted pathnames. However, if `pglob->gl_pathc` is 0, the content of `pglob->gl_pathv` is undefined.

The `globfree()` function returns no value.

If `glob()` terminates due to an error, it returns one of the non-zero constants defined in `<glob.h>`. The arguments `pglob->gl_pathc` and `pglob->gl_pathv` are still set as defined above.

ERRORS

No errors are defined.

EXAMPLES

One use of the GLOB_DOOFFS flag is by applications that build an argument list for use with `execv()`, `execve()` or `execvp()`. Suppose, for example, that an application wants to do the equivalent of:

```
ls -l *.c
```

but for some reason:

```
system("ls -l *.c")
```

is not acceptable. The application could obtain approximately the same result using the sequence:

```c
globbuf.gl_offs = 2;
glob ("*.c", GLOB_DOOFFS, NULL, &globbuf);
globbuf.gl_pathv[0] = "ls";
globbuf.gl_pathv[1] = "-l";
execvp ("ls", &globbuf.gl_pathv[0]);
```

Using the same example:

```
ls -l *.c *.h
```

could be approximately simulated using GLOB_APPEND as follows:

```c
globbuf.gl_offs = 2;
glob ("*.c", GLOB_DOOFFS, NULL, &globbuf);
glob ("*.h", GLOB_DOOFFS|GLOB_APPEND, NULL, &globbuf);
```

APPLICATION USAGE

This function is not provided for the purpose of enabling utilities to perform pathname expansion on their arguments, as this operation is performed by the shell, and utilities are explicitly not expected to redo this. Instead, it is provided for applications that need to do pathname expansion on strings obtained from other sources, such as a pattern typed by a user or read from a file.

If a utility needs to see if a pathname matches a given pattern, it can use `fnmatch()`.

Note that `gl_pathc` and `gl_pathv` have meaning even if `glob()` fails. This allows `glob()` to report partial results in the event of an error. However, if `gl_pathc` is 0, `gl_pathv` is unspecified even if `glob()` did not return an error.

The GLOB_NOCHECK option could be used when an application wants to expand a pathname if wildcards are specified, but wants to treat the pattern as just a string otherwise. The `sh` utility might use this for option-arguments, for example.
The new pathnames generated by a subsequent call with GLOB_APPEND are not sorted together with the previous pathnames. This mirrors the way that the shell handles pathname expansion when multiple expansions are done on a command line.

Applications that need tilde and parameter expansion should use \texttt{wordexp()}. 

\textbf{SEE ALSO} \texttt{execv()}, \texttt{fnmatch()}, \texttt{opendir()}, \texttt{readdir()}, \texttt{stat()}, \texttt{wordexp()}, \texttt{\textless glob.h\textgreater}, the XCU specification.

\textbf{CHANGE HISTORY}
First released in Issue 4.
Derived from the ISO POSIX-2 standard.
NAME
gmtime — convert time value to broken-down UTC time

SYNOPSIS
#include <time.h>
struct tm *gmtime(const time_t *timer);

DESCRIPTION
The gmtime() function converts the time in seconds since the Epoch pointed to by timer into a broken-down time, expressed as Coordinated Universal Time (UTC).

RETURN VALUE
The gmtime() function returns a pointer to a struct tm.

ERRORS
No errors are defined.

APPLICATION USAGE
The asctime(), ctime(), gmtime() and localtime() functions return values in one of two static objects: a broken-down time structure and an array of char. Execution of any of the functions may overwrite the information returned in either of these objects by any of the other functions.

SEE ALSO
asctime(), clock(), ctime(), difftime(), localtime(), mktime(), strftime(), strptime(), time(), utime(), <time.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:

• The type of argument timer is changed from time_t* to const time_t*.

Another change is incorporated as follows:

• In the APPLICATION USAGE section, the list of functions with which this function may interact is revised and the wording clarified.
NAME
grantpt — grant access to the slave pseudo-terminal device

SYNOPSIS

```c
#include <stdlib.h>

int grantpt(int fildes);
```

DESCRIPTION

The `grantpt()` function changes the mode and ownership of the slave pseudo-terminal device associated with its master pseudo-terminal counterpart. The `fildes` argument is a file descriptor that refers to a master pseudo-terminal device. The user ID of the slave is set to the real UID of the calling process and the group ID is set to an unspecified group ID. The permission mode of the slave pseudo-terminal is set to readable and writable by the owner, and writable by the group.

RETURN VALUE

Upon successful completion, `grantpt()` returns 0. Otherwise, it returns −1 and sets `errno` to indicate the error.

ERRORS

The `grantpt()` function may fail if:

- `[EBADF]` The `fildes` argument is not a valid open file descriptor.
- `[EINVAL]` The `fildes` argument is not associated with a master pseudo-terminal device.
- `[EACCES]` The corresponding slave pseudo-terminal device could not be accessed.

APPLICATION USAGE

The `grantpt()` function may also fail if the application has installed a signal handler to catch SIGCHLD signals.

SEE ALSO

`open()`, `ptsname()`, `unlockpt()`, `<stdlib.h>`.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
hcreate — create hash search tables

SYNOPSIS
EX
#include <search.h>

int hcreate(size_t nel);

DESCRIPTION
Refer to hsearch().

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:
• The type of argument nel is changed from unsigned to size_t.
NAME
hdestroy — destroy hash search tables

SYNOPSIS
EX
```
#include <search.h>

void hdestroy(void);
```

DESCRIPTION
Refer to hsearch().

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:
- The argument list is explicitly defined as void.
NAME
hsearch, hcreate, hdestroy — manage hash search tables

SYNOPSIS
#include <search.h>
ENTRY *hsearch (ENTRY item, ACTION action);
int hcreate(size_t nel);
void hdestroy(void);

DESCRIPTION
The hsearch() function is a hash-table search routine. It returns a pointer into a hash table indicating the location at which an entry can be found. The item argument is a structure of type ENTRY (defined in the <search.h> header) containing two pointers: item.key points to the comparison key (a char *), and item.data (a void *) points to any other data to be associated with that key. The comparison function used by hsearch() is strcmp(). The action argument is a member of an enumeration type ACTION indicating the disposition of the entry if it cannot be found in the table. ENTER indicates that the item should be inserted in the table at an appropriate point. FIND indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a null pointer.

The hcreate() function allocates sufficient space for the table, and must be called before hsearch() is used. The nel argument is an estimate of the maximum number of entries that the table will contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favourable circumstances.

The hdestroy() function disposes of the search table, and may be followed by another call to hcreate(). After the call to hdestroy(), the data can no longer be considered accessible.

RETURN VALUE
The hsearch() function returns a null pointer if either the action is FIND and the item could not be found or the action is ENTER and the table is full.

The hcreate() function returns 0 if it cannot allocate sufficient space for the table, and returns non-zero otherwise.

The hdestroy() function returns no value.

EXAMPLES
The following example will read in strings followed by two numbers and store them in a hash table, discarding duplicates. It will then read in strings and find the matching entry in the hash table and print it out.
#include <stdio.h>
#include <search.h>
#include <string.h>
struct info { /* this is the info stored in the table */
    int age, room; /* other than the key. */
};
#define NUM_EMPL 5000 /* # of elements in search table */
int main(void)
{
    char string_space[NUM_EMPL*20]; /* space to store strings */
    struct info info_space[NUM_EMPL]; /* space to store employee info*/
    char *str_ptr = string_space; /* next space in string_space */
    struct info *info_ptr = info_space; /* next space in info_space */
    ENTRY item;
    ENTRY *found_item; /* name to look for in table */
    char name_to_find[30];

    int i = 0;

    /* create table; no error checking is performed */
    (void) hcreate(NUM_EMPL);
    while (scanf("%s%d%d", str_ptr, &info_ptr->age,
                &info_ptr->room) != EOF && i++ < NUM_EMPL) {

        /* put information in structure, and structure in item */
        item.key = str_ptr;
        item.data = info_ptr;
        str_ptr += strlen(str_ptr) + 1;
        info_ptr++;

        /* put item into table */
        (void) hsearch(item, ENTER);
    }

    /* access table */
    item.key = name_to_find;
    while (scanf("%s", item.key) != EOF) {
        if (((found_item = hsearch(item, FIND)) != NULL) { 

            /* if item is in the table */
            (void)printf("found %s, age = %d, room = %d\n",
                         found_item->key,
                         ((struct info *)found_item->data)->age,
                         ((struct info *)found_item->data)->room);
        } else
            (void)printf("no such employee %s\n", name_to_find);
    }
    return 0;
}

ERRORS

The hsearch() and hcreate() functions may fail if:

[ENOMEM] Insufficient storage space is available.

APPLICATION USAGE

The hsearch() and hcreate() functions may use malloc() to allocate space.

SEE ALSO

bsearch(), lsearch(), malloc(), strcmp(), tsearch(), <search.h>.
CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated in this issue:

- In the **SYNOPSIS** section, the type of argument *nel* in the declaration of `hcreate()` is changed from `unsigned` to `size_t`, and the argument list is explicitly defined as `void` in the declaration of `hdestroy()`.

- In the **DESCRIPTION** section, the type of the comparison key is explicitly defined as `char *`, the type of `item.data` is explicitly defined as `void*`, and a statement is added indicating that `hsearch()` uses `strcmp()` as the comparison function.

- In the **EXAMPLES** section, the sample code is updated to use ISO C syntax.

- An **ERRORS** section is added and `[ENOMEM]` is defined as an error that may be returned by `hsearch()` and `hcreate()`.
NAME
hypot — Euclidean distance function

SYNOPSIS
#include <math.h>

double hypot(double x, double y);

DESCRIPTION
The hypot() function computes the length of the hypotenuse of a right-angled triangle:
\[ \sqrt{x^2 + y^2} \]

RETURN VALUE
Upon successful completion, hypot() returns the length of the hypotenuse of a right angled triangle with sides of length x and y.

If the result would cause overflow, HUGE_VAL is returned and errno may be set to [ERANGE].

If x or y is NaN, NaN is returned. and errno may be set to [EDOM].

If the correct result would cause underflow, 0 is returned and errno may be set to [ERANGE].

ERRORS
The hypot() function may fail if:
[EDOM] The value of x or y is NaN.
[ERANGE] The result overflows or underflows.

No other errors will occur.

APPLICATION USAGE
The hypot() function takes precautions against overflow during intermediate steps of the computation. If the calculated result would still overflow a double, then hypot() returns HUGE_VAL.

An application wishing to check for error situations should set errno to 0 before calling hypot(). If errno is non-zero on return, or the return value is HUGE_VAL or NaN, an error has occurred.

SEE ALSO
isnan(), sqrt(), <math.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:
• References to matherr() are removed.
• The RETURN VALUE and ERRORS sections are substantially rewritten to rationalise error handling in the mathematics functions.
NAME
iconv — code conversion function

SYNOPSIS

```c
size_t iconv(iconv_t cd, const char **inbuf, size_t *inbytesleft,
             char **outbuf, size_t *outbytesleft);
```

DESCRIPTION

The `iconv()` function converts the sequence of characters from one codeset, in the array specified by `inbuf`, into a sequence of corresponding characters in another codeset, in the array specified by `outbuf`. The codesets are those specified in the `iconv_open()` call that returned the conversion descriptor, `cd`. The `inbuf` argument points to a variable that points to the first character in the input buffer and `inbytesleft` indicates the number of bytes to the end of the buffer to be converted. The `outbuf` argument points to a variable that points to the first available byte in the output buffer and `outbytesleft` indicates the number of the available bytes to the end of the buffer.

For state-dependent encodings, the conversion descriptor `cd` is placed into its initial shift state by a call for which `inbuf` is a null pointer, or for which `inbuf` points to a null pointer. When `iconv()` is called in this way, and if `outbuf` is not a null pointer or a pointer to a null pointer, and `outbytesleft` points to a positive value, `iconv()` will place, into the output buffer, the byte sequence to change the output buffer to its initial shift state. If the output buffer is not large enough to hold the entire reset sequence, `iconv()` will fail and set `errno` to [E2BIG]. Subsequent calls with `inbuf` other than a null pointer or a pointer to a null pointer cause the conversion to take place from the current state of the conversion descriptor.

If a sequence of input bytes does not form a valid character in the specified codeset, conversion stops after the previous successfully converted character. If the input buffer ends with an incomplete character or shift sequence, conversion stops after the previous successfully converted bytes. If the output buffer is not large enough to hold the entire converted input, conversion stops just prior to the input bytes that would cause the output buffer to overflow. The variable pointed to by `inbuf` is updated to point to the byte following the last byte successfully used in the conversion. The value pointed to by `inbytesleft` is decremented to reflect the number of bytes still not converted in the input buffer. The variable pointed to by `outbuf` is updated to point to the byte following the last byte of converted output data. The value pointed to by `outbytesleft` is decremented to reflect the number of bytes still available in the output buffer. For state-dependent encodings, the conversion descriptor is updated to reflect the shift state in effect at the end of the last successfully converted byte sequence.

If `iconv()` encounters a character in the input buffer that is valid, but for which an identical character does not exist in the target codeset, `iconv()` performs an implementation-dependent conversion on this character.

RETURN VALUE

The `iconv()` function updates the variables pointed to by the arguments to reflect the extent of the conversion and returns the number of non-identical conversions performed. If the entire string in the input buffer is converted, the value pointed to by `inbytesleft` will be 0. If the input conversion is stopped due to any conditions mentioned above, the value pointed to by `inbytesleft` will be non-zero and `errno` is set to indicate the condition. If an error occurs `iconv()` returns `(size_t)-1` and sets `errno` to indicate the error.
**ERRORS**

The `iconv()` function will fail if:

- **[EILSEQ]** Input conversion stopped due to an input byte that does not belong to the input codeset.
- **[E2BIG]** Input conversion stopped due to lack of space in the output buffer.
- **[EINVAL]** Input conversion stopped due to an incomplete character or shift sequence at the end of the input buffer.

The `iconv()` function may fail if:

- **[EBADF]** The `cd` argument is not a valid open conversion descriptor.

**APPLICATION USAGE**

The `inbuf` argument indirectly points to the memory area which contains the conversion input data. The `outbuf` argument indirectly points to the memory area which is to contain the result of the conversion. The objects indirectly pointed to by `inbuf` and `outbuf` are not restricted to containing data that is directly representable in the ISO C language `char` data type. The type of `inbuf` and `outbuf`, `char **`, does not imply that the objects pointed to are interpreted as null-terminated C strings or arrays of characters. Any interpretation of a byte sequence that represents a character in a given character set encoding scheme is done internally within the code set converters. For example, the area pointed to indirectly by `inbuf` and/or `outbuf` can contain all zero octets that are not interpreted as string terminators but as coded character data according to the respective code set encoding scheme. The type of the data (`char`, `short int`, `long int`, and so on) read or stored in the objects is not specified, but may be inferred for both the input and output data by the converters determined by the `fromcode` and `tocode` arguments of `iconv_open()`.

Regardless of the data type inferred by the converter, the size of the remaining space in both input and output objects (the `inbytesleft` and `outbytesleft` arguments) is always measured in bytes.

For implementations that support the conversion of state-dependent encodings, the conversion descriptor must be able to accurately reflect the shift-state in effect at the end of the last successful conversion. It is not required that the conversion descriptor itself be updated, which would require it to be a pointer type. Thus, implementations are free to implement the descriptor as a handle (other than a pointer type) by which the conversion information can be accessed and updated.

**SEE ALSO**

`iconv_open()`, `iconv_close()`, `<iconv.h>`.

**CHANGE HISTORY**

First released in Issue 4.

Derived from the HP-UX manual.
iconv_close()  

NAME  
iconv_close — code conversion deallocation function  

SYNOPSIS  
EX  

```c  
#include <iconv.h>  

int iconv_close(iconv_t cd);  
```

DESCRIPTION  
The `iconv_close()` function deallocates the conversion descriptor `cd` and all other associated resources allocated by `iconv_open()`.

If a file descriptor is used to implement the type `iconv_t`, that file descriptor will be closed.

RETURN VALUE  
Upon successful completion, 0 is returned. Otherwise, −1 is returned and `errno` is set to indicate the error.

ERRORS  
The `iconv_close()` function may fail if:

- [EBADF] The conversion descriptor is invalid.

SEE ALSO  
`iconv()`, `iconv_open()`, `<iconv.h>`.

CHANGE HISTORY  
First released in Issue 4.

Derived from the HP-UX manual.
NAME
iconv_open — code conversion allocation function

SYNOPSIS
EX
#include <iconv.h>

iconv_t iconv_open(const char * tocode , const char * fromcode );

DESCRIPTION
The iconv_open() function returns a conversion descriptor that describes a conversion from the
codeset specified by the string pointed to by the fromcode argument to the codeset specified by
the string pointed to by the tocode argument. For state-dependent encodings, the conversion
descriptor will be in a codeset-dependent initial shift state, ready for immediate use with iconv().

Settings of fromcode and tocode and their permitted combinations are implementation-dependent.

A conversion descriptor remains valid in a process until that process closes it.

If a file descriptor is used to implement conversion descriptors, the FD_CLOEXEC flag will be
set; see <fcntl.h>.

RETURN VALUE
Upon successful completion, iconv_open() returns a conversion descriptor for use on subsequent
calls to iconv(). Otherwise iconv_open() returns (iconv_t)−1 and sets errno to indicate the error.

ERRORS
The iconv_open() function may fail if:
[EMFILE] [OPEN_MAX] files descriptors are currently open in the calling process.
[ENFILE] Too many files are currently open in the system.
[ENOMEM] Insufficient storage space is available.
[EINVAL] The conversion specified by fromcode and tocode is not supported by the
implementation.

APPLICATION USAGE
Some implementations of iconv_open() use malloc() to allocate space for internal buffer areas.
The iconv_open() function may fail if there is insufficient storage space to accommodate these
buffers.

Portable applications must assume that conversion descriptors are not valid after a call to one of
the exec functions.

SEE ALSO
iconv(), iconv_close(), <iconv.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the HP-UX manual.
NAME
ilogb - returns an unbiased exponent

SYNOPSIS
UX
#include <math.h>

int ilogb (double x)

DESCRIPTION
The ilogb() function returns the exponent part of x. Formally, the return value is the integral part of \log_r |x| as a signed integral value, for non-zero x, where r is the radix of the machine's floating point arithmetic.

The call ilogb(x) is equivalent to (int)logb(x).

RETURN VALUE
Upon successful completion, ilogb() returns the exponent part of x.

If x is 0 or NaN, then ilogb() returns INT_MIN. If x is ±Inf, then ilogb() returns INT_MAX.

ERRORS
No errors are defined.

SEE ALSO
logb(), <math.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
index — character string operations

SYNOPSIS
UX
#include <strings.h>

char *index(const char *s, int c);

DESCRIPTION
The index() function is identical to strchr().

RETURN VALUE
See strchr().

ERRORS
See strchr().

APPLICATION USAGE
For portability to implementations conforming to earlier versions of this document, strchr() is preferred over these functions.

SEE ALSO
strchr(), <strings.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
initstate()  X/OPEN UNIX  System Interfaces

NAME
initstate, random, setstate, srandom — pseudorandom number functions

SYNOPSIS

```c
#include <stdlib.h>

char *initstate(unsigned int seed, char *state, size_t size);
long random(void);
char *setstate(const char *state);
void srandom(unsigned int seed);
```

DESCRIPTION

The `random()` function uses a nonlinear additive feedback random-number generator employing a default state array size of 31 long integers to return successive pseudo-random numbers in the range from 0 to $2^{31}-1$. The period of this random-number generator is approximately $16 \times 2^{31}-1$. The size of the state array determines the period of the random-number generator. Increasing the state array size increases the period.

With 256 bytes of state information, the period of the random-number generator is greater than $2^{69}$.

Like `rand()`, `random()` produces by default a sequence of numbers that can be duplicated by calling `srandom()` with 1 as the seed.

The `srandom()` function initialises the current state array using the value of `seed`.

The `initstate()` and `setstate()` functions handle restarting and changing random-number generators. The `initstate()` function allows a state array, pointed to by the `state` argument, to be initialised for future use. The `size` argument, which specifies the size in bytes of the state array, is used by `initstate()` to decide what type of random-number generator to use; the larger the state array, the more random the numbers. Values for the amount of state information are 8, 32, 64, 128, and 256 bytes. Other values greater than 8 bytes are rounded down to the nearest one of these values. For values smaller than 8, `random()` uses a simple linear congruential random number generator. The `seed` argument specifies a starting point for the random-number sequence and provides for restarting at the same point. The `initstate()` function returns a pointer to the previous state information array.

If `initstate()` has not been called, then `random()` behaves as though `initstate()` had been called with `seed=1` and `size=128`.

If `initstate()` is called with `size<8`, then `random()` uses a simple linear congruential random number generator.

Once a state has been initialised, `setstate()` allows switching between state arrays. The array defined by the `state` argument is used for further random-number generation until `initstate()` is called or `setstate()` is called again. The `setstate()` function returns a pointer to the previous state array.

RETURN VALUE

The `random()` function returns the generated pseudo-random number.

The `srandom()` function returns no value.

Upon successful completion, `initstate()` and `setstate()` return a pointer to the previous state array. Otherwise, a null pointer is returned.
ERRORS
No errors are defined.

APPLICATION USAGE
After initialisation, a state array can be restarted at a different point in one of two ways:

- The `initstate()` function can be used, with the desired seed, state array, and size of the array.

- The `setstate()` function, with the desired state, can be used, followed by `srandom()` with the desired seed. The advantage of using both of these functions is that the size of the state array does not have to be saved once it is initialised.

Although some implementations of `random()` have written messages to standard error, such implementations do not conform to this document.

SEE ALSO
`drand48()`, `rand()`, `<stdlib.h>`.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME

insque, remque — insert or remove an element in a queue

SYNOPSIS

```c
#include <search.h>

void insque(void *element, void *pred);
void remque(void *element);
```

DESCRIPTION

The `insque()` and `remque()` functions manipulate queues built from doubly-linked lists. The queue can be either circular or linear. An application using `insque()` or `remque()` must define a structure in which the first two members of the structure are pointers to the same type of structure, and any further members are application-specific. The first member of the structure is a forward pointer to the next entry in the queue. The second member is a backward pointer to the previous entry in the queue. If the queue is linear, the queue is terminated with null pointers. The names of the structure and of the pointer members are not subject to any special restriction.

The `insque()` function inserts the element pointed to by `element` into a queue immediately after the element pointed to by `pred`.

The `remque()` function removes the element pointed to by `element` from a queue.

If the queue is to be used as a linear list, invoking `insque(&element, NULL)`, where `element` is the initial element of the queue, will initialise the forward and backward pointers of `element` to null pointers.

If the queue is to be used as a circular list, the application must initialise the forward pointer and the backward pointer of the initial element of the queue to the element’s own address.

RETURN VALUE

The `insque()` and `remque()` functions do not return a value.

ERRORS

No errors are defined.

APPLICATION USAGE

The historical implementations of these functions described the arguments as being of type `struct qelem *` rather than as being of type `void *` as defined here. In those implementations, `struct qelem` was commonly defined in `<search.h>` as:

```c
struct qelem {
    struct qelem  *q_forw;
    struct qelem  *q_back;
};
```

Applications using these functions, however, were never able to use this structure directly since it provided no room for the actual data contained in the elements. Most applications defined structures that contained the two pointers as the initial elements and also provided space for, or pointers to, the object’s data. Applications that used these functions to update more than one type of table also had the problem of specifying two or more different structures with the same name, if they literally used `struct qelem` as specified.

As described here, the implementations were actually expecting a structure type where the first two members were forward and backward pointers to structures. With C compilers that didn't provide function prototypes, applications used structures as specified in the DESCRIPTION above and the compiler did what the application expected.
If this method had been carried forward with an ISO C compiler and the historical function prototype, most applications would have to be modified to cast pointers to the structures actually used to be pointers to `struct qelem` to avoid compilation warnings. By specifying `void *` as the argument type, applications won’t need to change (unless they specifically referenced `struct qelem` and depended on it being defined in `<search.h>`).

SEE ALSO

`<search.h>`.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
ioctl — control device

SYNOPSIS
#include <stropts.h>

int ioctl(int fildes, int request, ... /* arg */);

DESCRIPTION
The ioctl() function performs a variety of control functions on STREAMS devices. For non-STREAMS devices, the functions performed by this call are unspecified. The request argument and an optional third argument (with varying type) are passed to and interpreted by the appropriate part of the STREAM associated with fildes.

The fildes argument is an open file descriptor that refers to a device.

The request argument selects the control function to be performed and will depend on the STREAMS device being addressed.

The arg argument represents additional information that is needed by this specific STREAMS device to perform the requested function. The type of arg depends upon the particular control request, but it is either an integer or a pointer to a device-specific data structure.

The ioctl() commands applicable to STREAMS, their arguments, and error statuses that apply to each individual command are described below.

The following ioctl() commands, with error values indicated, are applicable to all STREAMS files:

I_PUSH Pushes the module whose name is pointed to by arg onto the top of the current STREAM, just below the STREAM head. It then calls the open() function of the newly-pushed module.

The ioctl() function with the I_PUSH command will fail if:

[EINVAL] Invalid module name.

I_POP Removes the module just below the STREAM head of the STREAM pointed to by fildes. The arg argument should be 0 in an I_POP request.

The ioctl() function with the I_POP command will fail if:

[EINVAL] No module present in the STREAM.
[ENXIO] Hangup received on fildes.

I_LOOK Retrieves the name of the module just below the STREAM head of the STREAM pointed to by fildes, and places it in a character string pointed to by arg. The buffer pointed to by arg should be at least FMNAMESZ+1 bytes long, where FMNAMESZ is defined in <stropts.h>.

The ioctl() function with the I_LOOK command will fail if:

[EINVAL] No module present in the STREAM.
[ENXIO] Hangup received on fildes.

I_FLUSH This request flushes read and/or write queues, depending on the value of arg. Valid arg values are:
System Interfaces  

X/OPEN UNIX

**ioctl()**

- FLUSHR: Flush all read queues.
- FLUSHW: Flush all write queues.
- FLUSHRW: Flush all read and all write queues.

The `ioctl()` function with the I_FLUSH command will fail if:

- **[EINVAL]**: Invalid arg value.
- **[EAGAIN]** or **[ENOSR]**: Unable to allocate buffers for flush message.
- **[ENXIO]**: Hangup received on fildes.

1. **I_FLUSHBAND**
   Flushes a particular band of messages. The *arg* argument points to a `bandinfo` structure. The *bi_flag* member may be one of FLUSHR, FLUSHW, or FLUSHRW as described above. The *bi_pri* member determines the priority band to be flushed.

2. **I_SETSIG**
   Requests that the STREAMS implementation send the SIGPOLL signal to the calling process when a particular event has occurred on the STREAM associated with *fildes*. I_SETSIG supports an asynchronous processing capability in STREAMS. The value of *arg* is a bitmask that specifies the events for which the process should be signaled. It is the bitwise-OR of any combination of the following constants:

   - **S_RDNORM**: A normal (priority band set to 0) message has arrived at the head of a STREAM head read queue. A signal will be generated even if the message is of zero length.
   - **S_RDBAND**: A message with a non-zero priority band has arrived at the head of a STREAM head read queue. A signal will be generated even if the message is of zero length.
   - **S_INPUT**: A message, other than a high-priority message, has arrived at the head of a STREAM head read queue. A signal will be generated even if the message is of zero length.
   - **S_HIPRI**: A high-priority message is present on a STREAM head read queue. A signal will be generated even if the message is of zero length.
   - **S_OUTPUT**: The write queue for normal data (priority band 0) just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) normal data downstream.
   - **S_WRNORM**: Same as S_OUTPUT.
   - **S_WRBAND**: The write queue for a non-zero priority band just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) priority data downstream.
   - **S_MSG**: A STREAMS signal message that contains the SIGPOLL signal has reached the front of the STREAM head read queue.
   - **S_ERROR**: Notification of an error condition has reached the STREAM head.
S_HANGUP Notification of a hangup has reached the STREAM head.

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.

If \texttt{arg} is 0, the calling process will be unregistered and will not receive further SIGPOLL signals for the stream associated with \texttt{fildes}.

Processes that wish to receive SIGPOLL signals must explicitly register to receive them using \texttt{I_SETSIG}. If several processes register to receive this signal for the same event on the same STREAM, each process will be signaled when the event occurs.

The \texttt{ioctl()} function with the \texttt{I_SETSIG} command will fail if:

- \texttt{EINVAL} The value of \texttt{arg} is invalid.
- \texttt{EINVAL} The value of \texttt{arg} is 0 and the calling process is not registered to receive the SIGPOLL signal.
- \texttt{EAGAIN} There were insufficient resources to store the signal request.

\textbf{I_GETSIG} Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask in an \texttt{int} pointed to by \texttt{arg}, where the events are those specified in the description of \texttt{I_SETSIG} above.

The \texttt{ioctl()} function with the \texttt{I_GETSIG} command will fail if:

- \texttt{EINVAL} Process is not registered to receive the SIGPOLL signal.

\textbf{I_FIND} This request compares the names of all modules currently present in the STREAM to the name pointed to by \texttt{arg}, and returns 1 if the named module is present in the STREAM, or returns 0 if the named module is not present.

The \texttt{ioctl()} function with the \texttt{I_FIND} command will fail if:

- \texttt{EINVAL} \texttt{arg} does not contain a valid module name.

\textbf{I_PEEK} This request allows a process to retrieve the information in the first message on the STREAM head read queue without taking the message off the queue. It is analogous to \texttt{getmsg()} except that this command does not remove the message from the queue. The \texttt{arg} argument points to a \texttt{strpeek} structure.

The \texttt{maxlen} member in the \texttt{ctlbuf} and \texttt{databuf} structures must be set to the number of bytes of control information and/or data information, respectively, to retrieve. The \texttt{flags} member may be marked RS_HIPRI or 0, as described by \texttt{getmsg()}. If the process sets \texttt{flags} to RS_HIPRI, for example, \texttt{I_PEEK} will only look for a high-priority message on the STREAM head read queue.

\texttt{I_PEEK} returns 1 if a message was retrieved, and returns 0 if no message was found on the STREAM head read queue, or if the RS_HIPRI flag was set in \texttt{flags} and a high-priority message was not present on the STREAM head read queue. It does not wait for a message to arrive. On return, \texttt{ctlbuf} specifies information in the control buffer, \texttt{databuf} specifies information in the data buffer, and \texttt{flags} contains the value RS_HIPRI or 0.

\textbf{I_SRDOPT} Sets the read mode using the value of the argument \texttt{arg}. Read modes are described in \texttt{read()}. Valid \texttt{arg} flags are:
RNORM     Byte-stream mode, the default.
RMSGD     Message-discard mode.
RMSGN     Message-nondiscard mode.

The bitwise inclusive OR of RMSGD and RMSGN will return [EINVAL]. The bitwise inclusive OR of RNORM and either RMSGD or RMSGN will result in the other flag overriding RNORM which is the default.

In addition, treatment of control messages by the STREAM head may be changed by setting any of the following flags in arg:

RPROTNORM Fail read() with [EBADMSG] if a message containing a control part is at the front of the STREAM head read queue.
RPROTDAT  Deliver the control part of a message as data when a process issues a read().
RPROTDIS  Discard the control part of a message, delivering any data portion, when a process issues a read().

The ioctl() function with the I_SRDOPT command will fail if:

[EINVAL]     The arg argument is not valid.
I_GRDOPT     Returns the current read mode setting as, described above, in an int pointed to by the argument arg. Read modes are described in read().
I_NREAD     Counts the number of data bytes in the data part of the first message on the STREAM head read queue and places this value in the int pointed to by arg. The return value for the command is the number of messages on the STREAM head read queue. For example, if 0 is returned in arg, but the ioctl() return value is greater than 0, this indicates that a zero-length message is next on the queue.
I_FDINSERT Creates a message from specified buffer(s), adds information about another STREAM, and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below. The arg argument points to a strfdinsert structure.

The len member in the ctlbuf strbuf structure must be set to the size of a pointer plus the number of bytes of control information to be sent with the message. The fildes member specifies the file descriptor of the other STREAM, and the offset member, which must be suitably aligned for use as a pointer, specifies the offset from the start of the control buffer where I_FDINSERT will store a pointer whose interpretation is specific to the STREAM end. The len member in the databuf strbuf structure must be set to the number of bytes of data information to be sent with the message, or to 0 if no data part is to be sent.

The flags member specifies the type of message to be created. A normal message is created if flags is set to 0, and a high-priority message is created if flags is set to RS_HIPRI. For non-priority messages, I_FDINSERT will block if the STREAM write queue is full due to internal flow control conditions. For priority messages, I_FDINSERT does not block on this condition. For non-priority messages, I_FDINSERT does not block when the write queue is full and O_NONBLOCK is set. Instead, it fails and sets errno to [EAGAIN].
I_FDINSERT also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the STREAM, regardless of priority or whether O_NONBLOCK has been specified. No partial message is sent.

The ioctl() function with the I_FDINSERT command will fail if:

[EAGAIN] A non-priority message is specified, the O_NONBLOCK flag is set, and the STREAM write queue is full due to internal flow control conditions.

[EAGAIN] or [ENOSR] Buffers can not be allocated for the message that is to be created.

[EINVAL] One of the following:

- The fd member of the strfdinsert structure is not a valid, open STREAM file descriptor.
- The size of a pointer plus offset is greater than the len member for the buffer specified through ctlptr.
- The offset member does not specify a properly-aligned location in the data buffer.
- An undefined value is stored in flags.

[ENXIO] Hangup received on fd or fildes.

[ERANGE] The len member for the buffer specified through databuf does not fall within the range specified by the maximum and minimum packet sizes of the topmost STREAM module or the len member for the buffer specified through databuf is larger than the maximum configured size of the data part of a message; or the len member for the buffer specified through ctlbuf is larger than the maximum configured size of the control part of a message.

I_STR Constructs an internal STREAMS ioctl() message from the data pointed to by arg, and sends that message downstream.

This mechanism is provided to send ioctl() requests to downstream modules and drivers. It allows information to be sent with ioctl(), and returns to the process any information sent upstream by the downstream recipient. I_STR blocks until the system responds with either a positive or negative acknowledgement message, or until the request "times out" after some period of time. If the request times out, it fails with errno set to [ETIME].

At most, one I_STR can be active on a STREAM. Further I_STR calls will block until the active I_STR completes at the STREAM head. The default timeout interval for these requests is 15 seconds. The O_NONBLOCK flag has no effect on this call.

To send requests downstream, arg must point to a strioctl structure.

The ic_cmd member is the internal ioctl() command intended for a downstream module or driver and ic_timeout is the number of seconds (−1 = infinite, 0 = use implementation-dependent timeout interval, >0 = as specified) an I_STR request will wait for acknowledgement before timing out.
**ioctl()**

**ic_len** is the number of bytes in the data argument, and **ic_dp** is a pointer to the data argument. The **ic_len** member has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the process (the buffer pointed to by **ic_dp** should be large enough to contain the maximum amount of data that any module or the driver in the STREAM can return).

The STREAM head will convert the information pointed to by the **str_ioctl** structure to an internal **ioctl()** command message and send it downstream.

The **ioctl()** function with the I_STR command will fail if:

- **[EAGAIN]** or **[ENOSR]**: Unable to allocate buffers for the **ioctl()** message.
- **[EINVAL]**: The **ic_len** member is less than 0 or larger than the maximum configured size of the data part of a message, or **ic_timeout** is less than −1.
- **[ENXIO]**: Hangup received on **fildes**.
- **[ETIME]**: A downstream **ioctl()** timed out before acknowledgement was received.

An I_STR can also fail while waiting for an acknowledgement if a message indicating an error or a hangup is received at the STREAM head. In addition, an error code can be returned in the positive or negative acknowledgement message, in the event the **ioctl()** command sent downstream fails. For these cases, I_STR fails with **errno** set to the value in the message.

**I_SWROPT**

Sets the write mode using the value of the argument **arg**. Valid bit settings for **arg** are:

- **SNDZERO**: Send a zero-length message downstream when a **write()** of 0 bytes occurs. To not send a zero-length message when a **write()** of 0 bytes occurs, this bit must not be set in **arg** (for example, **arg** would be set to 0).

The **ioctl()** function with the I_SWROPT command will fail if:

- **[EINVAL]**: **arg** is not the above value.

**I_GWROPT**

Returns the current write mode setting, as described above, in the **int** that is pointed to by the argument **arg**.

**I_SENDFD**

**I_SENDFD** creates a new reference to the open file description associated with the file descriptor **arg**, and writes a message on the STREAMS-based pipe **fildes** containing this reference, together with the user ID and group ID of the calling process.

The **ioctl()** function with the I_SENDFD command will fail if:

- **[EAGAIN]**: The sending STREAM is unable to allocate a message block to contain the file pointer; or the read queue of the receiving STREAM head is full and cannot accept the message sent by **I_SENDFD**.
- **[EBADF]**: The **arg** argument is not a valid, open file descriptor.
- **[EINVAL]**: The **fildes** argument is not connected to a STREAM pipe.
ioctl()

X/OPEN UNIX

System Interfaces

I_RECVFD

Retrieves the reference to an open file description from a message written to a STREAMS-based pipe using the I_SENDFD command, and allocates a new file descriptor in the calling process that refers to this open file description. The arg argument is a pointer to an strrecvfd data structure as defined in <stropts.h>.

The fd member is a file descriptor. The uid and gid members are the effective user ID and effective group ID, respectively, of the sending process.

If O_NONBLOCK is not set I_RECVFD blocks until a message is present at the STREAM head. If O_NONBLOCK is set, I_RECVFD fails with errno set to [EAGAIN] if no message is present at the STREAM head.

If the message at the STREAM head is a message sent by an I_SENDFD, a new file descriptor is allocated for the open file descriptor referenced in the message. The new file descriptor is placed in the fd member of the strrecvfd structure pointed to by arg.

The ioctl() function with the I_RECVFD command will fail if:

- [EAGAIN] A message is not present at the STREAM head read queue and the O_NONBLOCK flag is set.
- [EBADMSG] The message at the STREAM head read queue is not a message containing a passed file descriptor.
- [EMFILE] The process has the maximum number of file descriptors currently open that it is allowed.
- [ENXIO] Hangup received on fildes.

I_LIST

This request allows the process to list all the module names on the STREAM, up to and including the topmost driver name. If arg is a null pointer, the return value is the number of modules, including the driver, that are on the STREAM pointed to by fildes. This lets the process allocate enough space for the module names. Otherwise, it should point to an str_list structure.

The sl_nmods member indicates the number of entries the process has allocated in the array. Upon return, the sl_modlist member of the str_list structure contains the list of module names, and the number of entries that have been filled into the sl_modlist array is found in the sl_nmods member (the number includes the number of modules including the driver). The return value from ioctl() is 0. The entries are filled in starting at the top of the STREAM and continuing downstream until either the end of the STREAM is reached, or the number of requested modules (sl_nmods) is satisfied.

The ioctl() function with the I_LIST command will fail if:

- [EINVAL] The sl_nmods member is less than 1.
- [EAGAIN] or [ENOSR] Unable to allocate buffers.

I_ATMARK

This request allows the process to see if the message at the head of the STREAM head read queue is marked by some module downstream. The arg argument determines how the checking is done when there may be multiple marked messages on the STREAM head read queue. It may take on the following values:
ANYMARK    Check if the message is marked.
LASTMARK   Check if the message is the last one marked on the queue.
The bitwise inclusive OR of the flags ANYMARK and LASTMARK is permitted.
The return value is 1 if the mark condition is satisfied and 0 otherwise.
The ioctl() function with the I_ATMARK command will fail if:
[EINVAL]   Invalid arg value.

I_CKBAND   Check if the message of a given priority band exists on the STREAM head read queue. This returns 1 if a message of the given priority exists, 0 if no message exists, or −1 on error. arg should be of type int.
The ioctl() function with the I_CKBAND command will fail if:
[EINVAL]   Invalid arg value.

I_GETBAND  Return the priority band of the first message on the STREAM head read queue in the integer referenced by arg.
The ioctl() function with the I_GETBAND command will fail if:
[ENODATA]  No message on the STREAM head read queue.

I_CANPUT   Check if a certain band is writable. arg is set to the priority band in question.
The return value is 0 if the band is flow-controlled, 1 if the band is writable, or −1 on error.
The ioctl() function with the I_CANPUT command will fail if:
[EINVAL]   Invalid arg value.

I_SETCLTIME This request allows the process to set the time the STREAM head will delay when a STREAM is closing and there is data on the write queues. Before closing each module or driver, if there is data on its write queue, the STREAM head will delay for the specified amount of time to allow the data to drain. If, after the delay, data is still present, they will be flushed. The arg argument is a pointer to an integer specifying the number of milliseconds to delay, rounded up to the nearest valid value. If I_SETCLTIME is not performed on a STREAM, an implementation-dependent default timeout interval is used.
The ioctl() function with the I_SETCLTIME command will fail if:
[EINVAL]   Invalid arg value.

I_GETCLTIME This request returns the close time delay in the integer pointed to by arg.

Multiplexed STREAMS Configurations
The following four commands are used for connecting and disconnecting multiplexed STREAMS configurations. These commands use an implementation-dependent default timeout interval.

I_LINK      Connects two STREAMs, where fildes is the file descriptor of the STREAM connected to the multiplexing driver, and arg is the file descriptor of the STREAM connected to another driver. The STREAM designated by arg gets connected below the multiplexing driver. I_LINK requires the multiplexing driver to send an acknowledgement message to the STREAM head regarding
The `ioctl()` function with the I_LINK command will fail if:

- [ENXIO] Hangup received on `fildes`.
- [ETIME] Time out before acknowledgement message was received at STREAM head.
- [EAGAIN] or [ENOSR] Unable to allocate STREAMS storage to perform the I_LINK.
- [EBADF] The `arg` argument is not a valid, open file descriptor.
- [EINVAL] The `fildes` argument does not support multiplexing; or `arg` is not a STREAM or is already connected downstream from a multiplexer; or the specified I_LINK operation would connect the STREAM head in more than one place in the multiplexed STREAM.

An I_LINK can also fail while waiting for the multiplexing driver to acknowledge the request, if a message indicating an error or a hangup is received at the STREAM head of `fildes`. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_LINK fails with `errno` set to the value in the message.

**I_UNLINK**

Disconnects the two STREAMs specified by `fildes` and `arg`. `fildes` is the file descriptor of the STREAM connected to the multiplexing driver. The `arg` argument is the multiplexer ID number that was returned by the I_LINK `ioctl()` command when a STREAM was connected downstream from the multiplexing driver. If `arg` is MUXID_ALL, then all STREAMs that were connected to `fildes` are disconnected. As in I_LINK, this command requires acknowledgement.

The `ioctl()` function with the I_UNLINK command will fail if:

- [ENXIO] Hangup received on `fildes`.
- [ETIME] Time out before acknowledgement message was received at STREAM head.
- [EAGAIN] or [ENOSR] Unable to allocate buffers for the acknowledgement message.
- [EINVAL] Invalid multiplexer ID number.

An I_UNLINK can also fail while waiting for the multiplexing driver to acknowledge the request if a message indicating an error or a hangup is received at the STREAM head of `fildes`. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_UNLINK fails with `errno` set to the value in the message.

**I_PLINK**

Creates a persistent connection between two STREAMs, where `fildes` is the file descriptor of the STREAM connected to the multiplexing driver, and `arg` is the file descriptor of the STREAM connected to another driver. This call creates a persistent connection which can exist even if the file descriptor `fildes` associated with the upper STREAM to the multiplexing driver is closed. The
STREAM designated by \textit{arg} gets connected via a persistent connection below the multiplexing driver. \texttt{I\_PLINK} requires the multiplexing driver to send an acknowledgement message to the STREAM head. This call returns a multiplexer ID number (an identifier that may be used to disconnect the multiplexer, see \texttt{I\_PUNLINK}) on success, and \texttt{−1} on failure.

The \texttt{ioctl()} function with the \texttt{I\_PLINK} command will fail if:

- \texttt{[ENXIO]} Hangup received on \texttt{fildes}.
- \texttt{[ETIME]} Time out before acknowledgement message was received at STREAM head.
- \texttt{[EAGAIN]} or \texttt{[ENOSR]} Unable to allocate STREAMS storage to perform the \texttt{I\_PLINK}.
- \texttt{[EBADF]} The \texttt{arg} argument is not a valid, open file descriptor.
- \texttt{[EINVAL]} The \texttt{fildes} argument does not support multiplexing; or \texttt{arg} is not a STREAM or is already connected downstream from a multiplexer; or the specified \texttt{I\_PLINK} operation would connect the STREAM head in more than one place in the multiplexed STREAM.

An \texttt{I\_PLINK} can also fail while waiting for the multiplexing driver to acknowledge the request, if a message indicating an error or a hangup is received at the STREAM head of \texttt{fildes}. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, \texttt{I\_PLINK} fails with \texttt{errno} set to the value in the message.

\texttt{I\_PUNLINK} Disconnects the two STREAMs specified by \texttt{fildes} and \texttt{arg} from a persistent connection. The \texttt{fildes} argument is the file descriptor of the STREAM connected to the multiplexing driver. The \texttt{arg} argument is the multiplexer ID number that was returned by the \texttt{I\_PLINK} \texttt{ioctl()} command when a STREAM was connected downstream from the multiplexing driver. If \texttt{arg} is \texttt{MUXID\_ALL} then all STREAMs which are persistent connections to \texttt{fildes} are disconnected. As in \texttt{I\_PLINK}, this command requires the multiplexing driver to acknowledge the request.

The \texttt{ioctl()} function with the \texttt{I\_PUNLINK} command will fail if:

- \texttt{[ENXIO]} Hangup received on \texttt{fildes}.
- \texttt{[ETIME]} Time out before acknowledgement message was received at STREAM head.
- \texttt{[EAGAIN]} or \texttt{[ENOSR]} Unable to allocate buffers for the acknowledgement message.
- \texttt{[EINVAL]} Invalid multiplexer ID number.

An \texttt{I\_PUNLINK} can also fail while waiting for the multiplexing driver to acknowledge the request if a message indicating an error or a hangup is received at the STREAM head of \texttt{fildes}. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, \texttt{I\_PUNLINK} fails with \texttt{errno} set to the value in the message.
RETURN VALUE
Upon successful completion, `ioctl()` returns a value other than −1 that depends upon the
STREAMS device control function. Otherwise, it returns −1 and sets `errno` to indicate the error.

ERRORS
Under the following general conditions, `ioctl()` will fail if:

- [EBADF]: The `fildes` argument is not a valid open file descriptor.
- [EINTR]: A signal was caught during the `ioctl()` operation.
- [EINVAL]: The STREAM or multiplexer referenced by `fildes` is linked (directly or
  indirectly) downstream from a multiplexer.

If an underlying device driver detects an error, then `ioctl()` will fail if:

- [EINVAL]: The `request` or `arg` argument is not valid for this device.
- [EIO]: Some physical I/O error has occurred.
- [ENOTTY]: The `fildes` argument is not associated with a STREAMS device that accepts
  control functions.
- [ENXIO]: The `request` and `arg` arguments are valid for this device driver, but the service
  requested can not be performed on this particular sub-device.
- [ENODEV]: The `fildes` argument refers to a valid STREAMS device, but the corresponding
  device driver does not support the `ioctl()` function.

If a STREAM is connected downstream from a multiplexer, any `ioctl()` command except
I_UNLINK and I_PUNLINK will set `errno` to [EINVAL].

APPLICATION USAGE
The implementation-defined timeout interval for STREAMS has historically been 15 seconds.

SEE ALSO
`close()`, `fcntl()`, `getmsg()`, `open()`, `pipe()`, `poll()`, `putmsg()`, `read()`,
`sigaction()`, `write()`, `<stropts.h>`,
Section 2.5 on page 35.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
  isalnum — test for alphanumeric character

SYNOPSIS
  #include <ctype.h>
  int isalnum(int c);

DESCRIPTION
  The isalnum() function tests whether c is a character of class alpha or digit in the program's current locale, see the XBD specification, Chapter 5, Locale.

  In all cases c is an int, the value of which must be representable as an unsigned char or must equal the value of the macro EOF. If the argument has any other value, the behaviour is undefined.

RETURN VALUE
  The isalnum() function returns non-zero if c is an alphanumeric character; otherwise it returns 0.

ERRORS
  No errors are defined.

APPLICATION USAGE
  To ensure application portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for character classification.

SEE ALSO
  isalpha(), iscntrl(), isdigit(), isgraph(), islower(), isprint(), ispunct(), isspace(), isupper(), isxdigit(), setlocale(), <ctype.h>, <stdio.h>, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY
  First released in Issue 1.

  Derived from Issue 1 of the SVID.

Issue 4
  The following change is incorporated in this issue:

    • The text of the DESCRIPTION and RETURN VALUE sections is revised, although there are no functional differences between this issue and Issue 3. Operation in the C locale is no longer described explicitly on this page.
NAME
    isalpha — test for alphabetic character

SYNOPSIS
    #include <ctype.h>
    int isalpha(int c);

DESCRIPTION
    The isalpha() function tests whether c is a character of class alpha in the program’s current locale, see the XBD specification, Chapter 5, Locale.
    
    In all cases c is an int, the value of which must be representable as an unsigned char or must equal the value of the macro EOF. If the argument has any other value, the behaviour is undefined.

RETURN VALUE
    The isalpha() function returns non-zero if c is an alphabetic character; otherwise it returns 0.

ERRORS
    No errors are defined.

APPLICATION USAGE
    To ensure application portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for character classification.

SEE ALSO
    isalnum(), iscntrl(), isdigit(), isgraph(), islower(), isprint(), ispunct(), isspace(), isupper(), isxdigit(), setlocale(), <ctype.h>, <stdio.h>, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY
    First released in Issue 1.
    
    Derived from Issue 1 of the SVID.

    Issue 4
    The following change is incorporated in this issue:
    
        • The text of the DESCRIPTION and RETURN VALUE sections is revised, although there are no functional differences between this issue and Issue 3. Operation in the C locale is no longer described explicitly on this page.
NAME
isascii — test for 7-bit US-ASCII character

SYNOPSIS
EX
#include <ctype.h>

int isascii(int c);

DESCRIPTION
The isascii() function tests whether c is a 7-bit US-ASCII character code.
The isascii() function is defined on all integer values.

RETURN VALUE
The isascii() function returns non-zero if c is a 7-bit US-ASCII character code between 0 and octal 0177 inclusive; otherwise it returns 0.

ERRORS
No errors are defined.

SEE ALSO
<ctype.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.
NAME
isastream — test a file descriptor

SYNOPSIS
UX
#include <stropts.h>
int isastream(int fildes);

DESCRIPTION
The isastream() function tests whether fildes, an open file descriptor, is associated with a
STREAMS-based file.

RETURN VALUE
Upon successful completion, isastream() returns 1 if fildes refers to a STREAMS-based file and 0 if
not. Otherwise, isastream() returns −1 and sets errno to indicate the error.

ERRORS
The isastream() function will fail if:

[EBADF] The fildes argument is not a valid open file descriptor.

SEE ALSO
<stropts.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
isatty( )

NAME
   isatty — test for a terminal device

SYNOPSIS
   #include <unistd.h>
   int isatty(int fildes);

DESCRIPTION
   The isatty( ) function tests whether fildes, an open file descriptor, is associated with a terminal
device.

RETURN VALUE
   The isatty( ) function returns 1 if fildes is associated with a terminal; otherwise it returns 0 and
   may set errno to indicate the error.

ERRORS
   The isatty( ) function may fail if:
   
   EX
      [EBADF] The fildes argument is not a valid open file descriptor.
      [ENOTTY] The fildes argument is not associated with a terminal.

APPLICATION USAGE
   The isatty( ) function does not necessarily indicate that a human being is available for interaction
   via fildes. It is quite possible that non-terminal devices are connected to the communications line.

SEE ALSO
   <unistd.h>.

CHANGE HISTORY
   First released in Issue 1.
   Derived from Issue 1 of the SVID.

   Issue 4
   The following changes are incorporated in this issue:
   
   • The header <unistd.h> is added to the SYNOPSIS section.
   • In the RETURN VALUE section, the sentence indicating that this function may set errno is
     marked as an extension.
   • The errors [EBADF] and [ENOTTY] are marked as extensions.
NAME
iscntrl — test for control character

SYNOPSIS
#include <ctype.h>
int iscntrl(int c);

DESCRIPTION
The iscntrl( ) function tests whether c is a character of class cntrl in the program's current locale, see the XBD specification, Chapter 5, Locale.

In all cases c is a type int, the value of which must be a character representable as an unsigned char or must equal the value of the macro EOF. If the argument has any other value, the behaviour is undefined.

RETURN VALUE
The iscntrl() function returns non-zero if c is a control character; otherwise it returns 0.

ERRORS
No errors are defined.

APPLICATION USAGE
To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for character classification.

SEE ALSO
isalnum(), isalpha(), isdigit(), isgraph(), islower(), isprint(), ispunct(), isspace(), isupper(), isxdigit(), setlocale(), <ctype.h>, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:

• The text of the DESCRIPTION and RETURN VALUE sections is revised, although there are no functional differences between this issue and Issue 3. Operation in the C locale is no longer described explicitly on this page.
NAME
isdigit — test for decimal digit

SYNOPSIS
#include <ctype.h>
int isdigit(int c);

DESCRIPTION
The isdigit() function tests whether c is a character of class digit in the program's current locale, see the XBD specification, Chapter 5, Locale.

In all cases c is an int, the value of which must be a character representable as an unsigned char or must equal the value of the macro EOF. If the argument has any other value, the behaviour is undefined.

RETURN VALUE
The isdigit() function returns non-zero if c is a decimal digit; otherwise it returns 0.

ERRORS
No errors are defined.

APPLICATION USAGE
To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for character classification.

SEE ALSO
isalnum(), isalpha(), iscntrl(), isgraph(), islower(), isprint(), ispunct(), isspace(), isupper(), isxdigit(), <ctype.h>.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:

• The text of the DESCRIPTION is revised, although there are no functional differences between this issue and Issue 3.
isgraph()  BASE  System Interfaces

NAME

isgraph — test for visible character

SYNOPSIS

#include <ctype.h>

int isgraph(int c);

DESCRIPTION

The isgraph() function tests whether c is a character of class graph in the program’s current locale, see the XBD specification, Chapter 5, Locale.

In all cases c is an int, the value of which must be a character representable as an unsigned char or must equal the value of the macro EOF. If the argument has any other value, the behaviour is undefined.

RETURN VALUE

The isgraph() function returns non-zero if c is a character with a visible representation; otherwise it returns 0.

ERRORS

No errors are defined.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for character classification.

SEE ALSO

isalnum(), isalpha(), iscntrl(), isdigit(), islower(), isprint(), ispunct(), isspace(), isupper(), isxdigit(), setlocale(), <ctype.h>, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following change is incorporated in this issue:

• The text of the DESCRIPTION and RETURN VALUE sections is revised, although there are no functional differences between this issue and Issue 3. Operation in the C locale is no longer described explicitly on this page.
NAME
islower — test for lower-case letter

SYNOPSIS
#include <ctype.h>

int islower(int c);

DESCRIPTION
The islower() function tests whether c is a character of class lower in the program's current locale, see the XBD specification, Chapter 5, Locale.

In all cases c is an int, the value of which must be a character representable as an unsigned char or must equal the value of the macro EOF. If the argument has any other value, the behaviour is undefined.

RETURN VALUE
The islower() function returns non-zero if c is a lower-case letter; otherwise it returns 0.

ERRORS
No errors are defined.

APPLICATION USAGE
To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for character classification.

SEE ALSO
isalnum(), isalpha(), iscntrl(), isdigit(), isgraph(), isprint(), ispunct(), isspace(), isupper(), isxdigit(), setlocale(), <ctype.h>, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:

- The text of the DESCRIPTION and RETURN VALUE sections is revised, although there are no functional differences between this issue and Issue 3. Operation in the C locale is no longer described explicitly on this page.
NAME
isnan — test for NaN

SYNOPSIS
EX
#include <math.h>

int isnan(double x);

DESCRIPTION
The isnan() function tests whether x is NaN.

RETURN VALUE
The isnan() function returns non-zero if x is NaN. Otherwise, 0 is returned.

ERRORS
No errors are defined.

APPLICATION USAGE
On systems not supporting NaN values, isnan() always returns 0.

SEE ALSO
<math.h>.

CHANGE HISTORY
First released in Issue 3.

Issue 4
The following change is incorporated in this issue:

• The words “not supporting NaN” are added to the APPLICATION USAGE section.
NAME
isprint — test for printing character

SYNOPSIS
#include <ctype.h>
int isprint(int c);

DESCRIPTION
The isprint() function tests whether c is a character of class print in the program's current locale, see the XBD specification, Chapter 5, Locale.

In all cases c is an int, the value of which must be a character representable as an unsigned char or must equal the value of the macro EOF. If the argument has any other value, the behaviour is undefined.

RETURN VALUE
The isprint() function returns non-zero if c is a printing character; otherwise it returns 0.

ERRORS
No errors are defined.

APPLICATION USAGE
To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for character classification.

SEE ALSO
isalnum(), isalpha(), iscntrl(), isdigit(), isgraph(), islower(), ispunct(), isspace(), isupper(), isxdigit(), setlocale(), <ctype.h>, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:

- The text of the DESCRIPTION and RETURN VALUE sections is revised, although there are no functional differences between this issue and Issue 3. Operation in the C locale is no longer described explicitly on this page.
NAME
ispunct — test for punctuation character

SYNOPSIS
#include <ctype.h>
int ispunct(int c);

DESCRIPTION
The ispunct() function tests whether c is a character of class punct in the program’s current locale, see the XBD specification, Chapter 5, Locale.

In all cases c is an int, the value of which must be a character representable as an unsigned char or must equal the value of the macro EOF. If the argument has any other value, the behaviour is undefined.

RETURN VALUE
The ispunct() function returns non-zero if c is a punctuation character; otherwise it returns 0.

ERRORS
No errors are defined.

APPLICATION USAGE
To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for character classification.

SEE ALSO
isalnum(), isalpha(), iscntrl(), isdigit(), isgraph(), islower(), isprint(), isspace(), isupper(), isxdigit(), setlocale(), <ctype.h>, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:

- The text of the DESCRIPTION and RETURN VALUE sections is revised, although there are no functional differences between this issue and Issue 3. Operation in the C locale is no longer described explicitly on this page.
NAME
isspace — test for white-space character

SYNOPSIS
#include <ctype.h>
int isspace(int c);

DESCRIPTION
The isspace() function tests whether \( c \) is a character of class space in the program's current locale, see the XBD specification, Chapter 5, Locale.

In all cases \( c \) is an int, the value of which must be a character representable as an unsigned char or must equal the value of the macro EOF. If the argument has any other value, the behaviour is undefined.

RETURN VALUE
The isspace() function returns non-zero if \( c \) is a white-space character; otherwise it returns 0.

ERRORS
No errors are defined.

APPLICATION USAGE
To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for character classification.

SEE ALSO
isalnum(), isalpha(), iscntrl(), isdigit(), isgraph(), islower(), isprint(), ispunct(), isupper(), isxdigit(), setlocale(), <ctype.h>, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:

- The text of the DESCRIPTION and RETURN VALUE sections is revised, although there are no functional differences between this issue and Issue 3. Operation in the C locale is no longer described explicitly on this page.
NAME
isupper — test for upper-case letter

SYNOPSIS
#include <ctype.h>
int isupper(int c);

DESCRIPTION
The isupper() function tests whether c is a character of class upper in the program's current locale, see the XBD specification, Chapter 5, Locale.

In all cases c is an int, the value of which must be a character representable as an unsigned char or must equal the value of the macro EOF. If the argument has any other value, the behaviour is undefined.

RETURN VALUE
The isupper() function returns non-zero if c is an upper-case letter; otherwise it returns 0.

ERRORS
No errors are defined.

APPLICATION USAGE
To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for character classification.

SEE ALSO
isalnum(), isalpha(), iscntrl(), isdigit(), isgraph(), islower(), isprint(), ispunct(), isspace(), isxdigit(), setlocale(), <ctype.h>, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:

- The text of the DESCRIPTION and RETURN VALUE sections is revised, although there are no functional differences between this issue and Issue 3. Operation in the C locale is no longer described explicitly on this page.
NAME

iswalnum — test for an alphanumeric wide-character code

SYNOPSIS

WP

```c
#include <wchar.h>

int iswalnum(wint_t wc);
```

DESCRIPTION

The `iswalnum()` function tests whether `wc` is a wide-character code representing a character of class `alpha` or `digit` in the program's current locale, see the XBD specification, Chapter 5, Locale.

In all cases `wc` is a `wint_t`, the value of which must be a wide-character code corresponding to a valid character in the current locale or must equal the value of the macro `WEOF`. If the argument has any other value, the behaviour is undefined.

RETURN VALUE

The `iswalnum()` function returns non-zero if `wc` is an alphanumeric wide-character code; otherwise it returns 0.

ERRORS

No errors are defined.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for classification of wide-character codes.

SEE ALSO

`iswalpha()`, `iswcntrl()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `<wchar.h>`, `<stdio.h>`, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY

First released as a World-wide Portability Interface in Issue 4.

Derived from the MSE working draft.
NAME
iswalpha — test for an alphabetic wide-character code

SYNOPSIS
WP
#include <wchar.h>

int iswalpha(wint_t wc);

DESCRIPTION
The iswalpha() function tests whether wc is a wide-character code representing a character of class alpha in the program's current locale, see the XBD specification, Chapter 5, Locale.

In all cases wc is a wint_t, the value of which must be a wide-character code corresponding to a valid character in the current locale or must equal the value of the macro WEOF. If the argument has any other value, the behaviour is undefined.

RETURN VALUE
The iswalpha() function returns non-zero if wc is an alphabetic wide-character code; otherwise it returns 0.

ERRORS
No errors are defined.

APPLICATION USAGE
To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for classification of wide-character codes.

SEE ALSO
iswalnum(), iswcntrl(), iswdigit(), iswgraph(), iswlower(), iswprint(), iswpunct(), iswspace(), iswupper(), iswxdigit(), setlocale(), <wchar.h>, <stdio.h>, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME

iswcntrl — test for a control wide-character code

SYNOPSIS

WP

```c
#include <wchar.h>

int iswcntrl(wint_t wc);
```

DESCRIPTION

The `iswcntrl()` function tests whether `wc` is a wide-character code representing a character of class `control` in the program's current locale, see the XBD specification, Chapter 5, Locale.

In all cases `wc` is a `wint_t`, the value of which must be a wide-character code corresponding to a valid character in the current locale or must equal the value of the macro `WEOF`. If the argument has any other value, the behaviour is undefined.

RETURN VALUE

The `iswcntrl()` function returns non-zero if `wc` is a control wide-character code; otherwise it returns 0.

ERRORS

No errors are defined.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for classification of wide-character codes.

SEE ALSO

`iswalnum()`, `iswalpha()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `<wchar.h>`, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY

First released in Issue 4.

Derived from the MSE working draft.
NAME

iswctype - test character for specified class

SYNOPSIS

WP

```c
#include <wchar.h>

int iswctype(wint_t wc, wctype_t charclass);
```

DESCRIPTION

The `iswctype()` function determines whether the wide-character code `wc` has the character class `charclass`, returning true or false. The `iswctype()` function is defined on WEOF and wide-character codes corresponding to the valid character encodings in the current locale. If the `wc` argument is not in the domain of the function, the result is undefined. If the value of `charclass` is invalid (that is, not obtained by a call to `wctype()` or `charclass` is invalidated by a subsequent call to `setlocale()` that has affected category LC_CTYPE) the result is implementation-dependent.

RETURN VALUE

The `iswctype()` function returns 0 for false and non-zero for true.

ERRORS

No errors are defined.

APPLICATION USAGE

The twelve strings — "alnum", "alpha", "blank", "cntrl", "digit", "graph", "lower", "print", "punct", "space", "upper" and "xdigit" — are reserved for the standard character classes. In the table below, the functions in the left column are equivalent to the functions in the right column.

- `iswalnum(wc)` is `iswctype(wc, wctype("alnum"))`
- `iswalpha(wc)` is `iswctype(wc, wctype("alpha"))`
- `iswcntrl(wc)` is `iswctype(wc, wctype("cntrl"))`
- `iswdigit(wc)` is `iswctype(wc, wctype("digit"))`
- `iswgraph(wc)` is `iswctype(wc, wctype("graph"))`
- `iswlower(wc)` is `iswctype(wc, wctype("lower"))`
- `iswprint(wc)` is `iswctype(wc, wctype("print"))`
- `iswpunct(wc)` is `iswctype(wc, wctype("punct"))`
- `iswspace(wc)` is `iswctype(wc, wctype("space"))`
- `iswupper(wc)` is `iswctype(wc, wctype("upper"))`
- `iswxdigit(wc)` is `iswctype(wc, wctype("xdigit"))`

**Note:** The call:

```c
iswctype(wc, wctype("blank"))
```

does not have an equivalent `isw*()` function.

SEE ALSO

- `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswspace()`, `iswupper()`, `iswxdigit()`, `wctype()`, `<wchar.h>`.

CHANGE HISTORY

First released as World-wide Portability Interfaces in Issue 4.

Derived from a proposal in the UniForum Technical Subcommittee on Internationalization and the MSE working draft.
NAME

iswdigit — test for a decimal digit wide-character code

SYNOPSIS

WP

```c
#include <wchar.h>

int iswdigit(wint_t wc);
```

DESCRIPTION

The `iswdigit()` function tests whether `wc` is a wide-character code representing a character of class `digit` in the program's current locale, see the XBD specification, Chapter 5, Locale.

In all cases `wc` is a `wint_t`, the value of which must be a wide-character code corresponding to a valid character in the current locale or must equal the value of the macro WEOF. If the argument has any other value, the behaviour is undefined.

RETURN VALUE

The `iswdigit()` function returns non-zero if `wc` is a decimal digit wide-character code; otherwise it returns 0.

ERRORS

No errors are defined.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for classification of wide-character codes.

SEE ALSO

`iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswspace()`, `iswupper()`, `iswxdigit()`, `<wchar.h>.

CHANGE HISTORY

First released in Issue 4.

Derived from the MSE working draft.
NAME

iswgraph — test for a visible wide-character code

SYNOPSIS

WP

```c
#include <wchar.h>

int iswgraph(wint_t wc);
```

DESCRIPTION

The `iswgraph()` function tests whether `wc` is a wide-character code representing a character of class `graph` in the program’s current locale, see the XBD specification, Chapter 5, Locale.

In all cases `wc` is a `wint_t`, the value of which must be a wide-character code corresponding to a valid character in the current locale or must equal the value of the macro `WEOF`. If the argument has any other value, the behaviour is undefined.

RETURN VALUE

The `iswgraph()` function returns non-zero if `wc` is a wide-character code with a visible representation; otherwise it returns 0.

ERRORS

No errors are defined.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for classification of wide-character codes.

SEE ALSO

`iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswdigit()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `<wchar.h>`, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY

First released in Issue 4.

Derived from the MSE working draft.
**NAME**
iswlower — test for a lower-case letter wide-character code

**SYNOPSIS**
```c
#include <wchar.h>

int iswlower(wint_t wc);
```

**DESCRIPTION**
The `iswlower()` function tests whether `wc` is a wide-character code representing a character of class `lower` in the program's current locale, see the XBD specification, Chapter 5, Locale.

In all cases `wc` is a `wint_t`, the value of which must be a wide-character code corresponding to a valid character in the current locale or must equal the value of the macro `WEOF`. If the argument has any other value, the behaviour is undefined.

**RETURN VALUE**
The `iswlower()` function returns non-zero if `wc` is a lower-case letter wide-character code; otherwise it returns 0.

**ERRORS**
No errors are defined.

**APPLICATION USAGE**
To ensure applications portability, especially across natural languages, only this function and those listed in the **SEE ALSO** section should be used for classification of wide-character codes.

**SEE ALSO**
iswalnum(), iswalpha(), iswcntrl(), iswdigit(), iswgraph(), iswprint(), iswpunct(), iswspace(), iswupper(), iswxdigit(), setlocale(), `<wchar.h>`, the XBD specification, Chapter 5, Locale.

**CHANGE HISTORY**
First released in Issue 4.
Derived from the MSE working draft.
NAME

   iswprint — test for a printing wide-character code

SYNOPSIS

   
   #include <wchar.h>
   
   int iswprint(wint_t wc);

DESCRIPTION

   The iswprint() function tests whether wc is a wide-character code representing a character of class print in the program's current locale, see the XBD specification, Chapter 5, Locale.

   In all cases wc is a wint_t, the value of which must be a wide-character code corresponding to a valid character in the current locale or must equal the value of the macro WEOF. If the argument has any other value, the behaviour is undefined.

RETURN VALUE

   The iswprint() function returns non-zero if wc is a printing wide-character code; otherwise it returns 0.

ERRORS

   No errors are defined.

APPLICATION USAGE

   To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for classification of wide-character codes.

SEE ALSO

   iswalnum(), iswalpha(), iswcntrl(), iswdigit(), iswgraph(), iswlower(), iswpunct(), iswspace(), iswupper(), iswxdigit(), setlocale(), <wchar.h>, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY

   First released in Issue 4.

   Derived from the MSE working draft.
NAME

iswpunct — test for a punctuation wide-character code

SYNOPSIS

WP

```c
#include <wchar.h>

int iswpunct(wint_t wc);
```

DESCRIPTION

The `iswpunct()` function tests whether `wc` is a wide-character code representing a character of class `punct` in the program's current locale, see the XBD specification, Chapter 5, Locale.

In all cases `wc` is a `wint_t`, the value of which must be a wide-character code corresponding to a valid character in the current locale or must equal the value of the macro `WEOF`. If the argument has any other value, the behaviour is undefined.

RETURN VALUE

The `iswpunct()` function returns non-zero if `wc` is a punctuation wide-character code; otherwise it returns 0.

ERRORS

No errors are defined.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for classification of wide-character codes.

SEE ALSO

`iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `<wchar.h>`, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY

First released in Issue 4.

Derived from the MSE working draft.
NAME

iswspace — test for a white-space wide-character code

SYNOPSIS

WP #include <wchar.h>

int iswspace(wint_t wc);

DESCRIPTION

The iswspace() function tests whether wc is a wide-character code representing a character of class space in the program's current locale, see the XBD specification, Chapter 5, Locale.

In all cases wc is a wint_t, the value of which must be a wide-character code corresponding to a valid character in the current locale or must equal the value of the macro WEOF. If the argument has any other value, the behaviour is undefined.

RETURN VALUE

The iswspace() function returns non-zero if wc is a white-space wide-character code; otherwise it returns 0.

ERRORS

No errors are defined.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for classification of wide-character codes.

SEE ALSO

iswalnum(), iswalpha(), iswcntrl(), iswdigit(), iswgraph(), iswlower(), iswprint(), iswpunct(), iswupper(), iswxdigit(), setlocale(), <wchar.h>, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY

First released in Issue 4.

Derived from the MSE working draft.
NAME
iswupper — test for an upper-case letter wide-character code

SYNOPSIS
WP
#include <wchar.h>

int iswupper(wint_t wc);

DESCRIPTION
The iswupper() function tests whether wc is a wide-character code representing a character of class upper in the program's current locale, see the XBD specification, Chapter 5, Locale.

In all cases wc is a wint_t, the value of which must be a wide-character code corresponding to a valid character in the current locale or must equal the value of the macro WEOF. If the argument has any other value, the behaviour is undefined.

RETURN VALUE
The iswupper() function returns non-zero if wc is an upper-case letter wide-character code; otherwise it returns 0.

ERRORS
No errors are defined.

APPLICATION USAGE
To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for classification of wide-character codes.

SEE ALSO
iswalnum(), iswalpha(), iswcntrl(), iswdigit(), iswgraph(), iswlower(), iswprint(), iswpunct(), iswspace(), iswxdigit(), setlocale(), <wchar.h>, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
iswxdigit — test for a hexadecimal digit wide-character code

SYNOPSIS

```c
#include <wchar.h>

int iswxdigit(wint_t wc);
```

DESCRIPTION
The `iswxdigit()` function tests whether `wc` is a wide-character code representing a character of class `xdigit` in the program's current locale, see the XBD specification, Chapter 5, Locale.

In all cases `wc` is a `wint_t`, the value of which must be a wide-character code corresponding to a valid character in the current locale or must equal the value of the macro `WEOF`. If the argument has any other value, the behaviour is undefined.

RETURN VALUE
The `iswxdigit()` function returns non-zero if `wc` is a hexadecimal digit wide-character code; otherwise it returns 0.

ERRORS
No errors are defined.

APPLICATION USAGE
To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for classification of wide-character codes.

SEE ALSO
`iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswspace()`, `iswupper()`, `setlocale()`, `<wchar.h>`.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
isxdigit — test for hexadecimal digit

SYNOPSIS
#include <ctype.h>
int isxdigit(int c);

DESCRIPTION
The isxdigit() function tests whether c is a character of class xdigit in the program's current locale, see the XBD specification, Chapter 5, Locale.

In all cases c is an int, the value of which must be a character representable as an unsigned char or must equal the value of the macro EOF. If the argument has any other value, the behaviour is undefined.

RETURN VALUE
The isxdigit() function returns non-zero if c is a hexadecimal digit; otherwise it returns 0.

ERRORS
No errors are defined.

APPLICATION USAGE
To ensure applications portability, especially across natural languages, only this function and those listed in the SEE ALSO section should be used for character classification.

SEE ALSO
isalnum(), isalpha(), iscntrl(), isdigit(), isgraph(), islower(), isprint(), ispunct(), isspace(), isupper(), <ctype.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:

• The text of the DESCRIPTION section is revised, although there are no functional differences between this issue and Issue 3.
**NAME**

j0, j1, jn — Bessel functions of the first kind

**SYNOPSIS**

```c
#include <math.h>

double j0(double x);
double j1(double x);
double jn(int n, double x);
```

**DESCRIPTION**

The j0(), j1() and jn() functions compute Bessel functions of x of the first kind of orders 0, 1 and n respectively.

**RETURN VALUE**

Upon successful completion, j0(), j1() and jn() return the relevant Bessel value of x of the first kind.

If the x argument is too large in magnitude, 0 is returned and errno may be set to [ERANGE].

If x is NaN, NaN is returned and errno may be set to [EDOM].

If the correct result would cause underflow, 0 is returned and errno may be set to [ERANGE].

**ERRORS**

The j0(), j1() and jn() functions may fail if:

- [EDOM] The value of x is NaN.
- [ERANGE] The value of x was too large in magnitude, or underflow occurred.

No other errors will occur.

**APPLICATION USAGE**

An application wishing to check for error situations should set errno to 0 before calling j0(), j1() or jn(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

**SEE ALSO**

isnan(), y0(), <math.h>.

**CHANGE HISTORY**

First released in Issue 1.

Derived from Issue 1 of the SVID.

**Issue 4**

The following changes are incorporated in this issue:

- References to matherr() are removed.

- The RETURN VALUE and ERRORS sections are substantially rewritten to rationalise error handling in the mathematics functions.
NAME
jrand48 — generate uniformly distributed pseudo-random long signed integers

SYNOPSIS
EX
```c
#include <stdlib.h>

long int jrand48(unsigned short int xsubi[3]);
```

DESCRIPTION
Refer to `drand48()`.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated this issue:
- The header `<stdlib.h>` is added to the SYNOPSIS section.
- The word `long` is replaced by the words `long int` in the SYNOPSIS section.
NAME
kill — send a signal to a process or a group of processes

SYNOPSIS

```
#include <sys/types.h>
#include <signal.h>

int kill(pid_t pid, int sig);
```

DESCRIPTION

The `kill()` function will send a signal to a process or a group of processes specified by `pid`. The signal to be sent is specified by `sig` and is either one from the list given in `<signal.h>` or 0. If `sig` is 0 (the null signal), error checking is performed but no signal is actually sent. The null signal can be used to check the validity of `pid`.

`-_POSIX_SAVED_IDS` will be defined on all XSI-conformant systems, and for a process to have permission to send a signal to a process designated by `pid`, the real or effective user ID of the sending process must match the real or saved set-user-ID of the receiving process, unless the sending process has appropriate privileges.

If `pid` is greater than 0, `sig` will be sent to the process whose process ID is equal to `pid`.

If `pid` is 0, `sig` will be sent to all processes (excluding an unspecified set of system processes) whose process group ID is equal to the process group ID of the sender, and for which the process has permission to send a signal.

If `pid` is negative, but not –1, `sig` will be sent to all processes (excluding an unspecified set of system processes) whose process group ID is equal to the absolute value of `pid`, and for which the process has permission to send a signal.

If the value of `pid` causes `sig` to be generated for the sending process, and if `sig` is not blocked, either `sig` or at least one pending unblocked signal will be delivered to the sending process before `kill()` returns.

The user ID tests described above will not be applied when sending SIGCONT to a process that is a member of the same session as the sending process.

An implementation that provides extended security controls may impose further implementation-dependent restrictions on the sending of signals, including the null signal. In particular, the system may deny the existence of some or all of the processes specified by `pid`.

The `kill()` function is successful if the process has permission to send `sig` to any of the processes specified by `pid`. If `kill()` fails, no signal will be sent.

RETURN VALUE

Upon successful completion, 0 is returned. Otherwise, –1 is returned and `errno` is set to indicate the error.

ERRORS

The `kill()` function will fail if:

- `[EINVAL]` The value of the `sig` argument is an invalid or unsupported signal number.
- `[EPERM]` The process does not have permission to send the signal to any receiving process.
- `[ESRCH]` No process or process group can be found corresponding to that specified by `pid`.

X/Open CAE Specification (1994)
SEE ALSO

getpid(), raise(), setsid(), sigaction(), <signal.h>, <sys/types.h>.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following change is incorporated for alignment with the FIPS requirements:

• In the DESCRIPTION section, the second paragraph is reworded to indicate that the saved set-user-ID of the calling process will be checked in place of its effective user ID. This functionality is marked as an extension.

Other changes are incorporated as follows:

• The header <sys/types.h> is now marked as optional (OH); this header need not be included on XSI-conformant systems.

• The DESCRIPTION section is clarified in various places.
NAME
killpg — send a signal to a process group

SYNOPSIS

```c
#include <signal.h>

int killpg(pid_t pgrp, int sig);
```

DESCRIPTION

The `killpg()` function sends the signal specified by `sig` to the process group specified by `pgrp`.

If `pgrp` is greater than 1, `killpg(pgrp, sig)` is equivalent to `kill(-pgrp, sig)`. If `pgrp` is less than or equal to 1, the behaviour of `killpg()` is undefined.

RETURN VALUE

Refer to `kill()`.

ERRORS

Refer to `kill()`.

SEE ALSO

`getpgid()`, `getpid()`, `kill()`, `raise()`, `<signal.h>`.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
l64a — convert 32-bit integer to radix-64 ASCII string

SYNOPSIS
UX
#include <stdlib.h>

char *l64a(long value);

DESCRIPTION
Refer to a64l().

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
labs — return long integer absolute value

SYNOPSIS
#include <stdlib.h>
long int labs(long int i);

DESCRIPTION
The labs() function computes the absolute value of its long integer operand, i. If the result cannot be represented, the behaviour is undefined.

RETURN VALUE
The labs() function returns the absolute value of its long integer operand.

ERRORS
No errors are defined.

SEE ALSO
abs(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the ISO C standard.
NAME
Ichown — change owner and group of a file

SYNOPSIS
UX
#include <unistd.h>

int lchown(const char *path, uid_t owner, gid_t group);

DESCRIPTION
The lchown() function has the same effect as chown() except in the case where the named file is a symbolic link. In this case lchown() changes the ownership of the symbolic link file itself, while chown() changes the ownership of the file or directory to which the symbolic link refers.

RETURN VALUE
Upon successful completion, lchown() returns 0. Otherwise, it returns −1 and sets errno to indicate an error.

ERRORS
The lchown() function will fail if:
[EACCESE] Search permission is denied on a component of the path prefix of path.
[EINVAL] The owner or group id is not a value supported by the implementation.
[ENAMETOOOLONG] The length of a pathname exceeds {PATH_MAX}, or pathname component is longer than {NAME_MAX}.
[ENOENT] A component of path does not name an existing file or path is an empty string.
[ENOTDIR] A component of the path prefix of path is not a directory.
[EOPNOTSUPP] The path argument names a symbolic link and the implementation does not support setting the owner or group of a symbolic link.
[ELOOP] Too many symbolic links were encountered in resolving path.
[EPERM] The effective user ID does not match the owner of the file and the process does not have appropriate privileges.
[EROFS] The file resides on a read-only file system.

The lchown() function may fail if:
[EIO] An I/O error occurred while reading or writing to the file system.
[EINTR] A signal was caught during execution of the function.
[ENAMETOOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

APPLICATION USAGE
On implementations which support symbolic links as directory entries rather than files, lchown() may fail.

SEE ALSO
chown(), symlink(), <unistd.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
lcong48 — seed uniformly distributed pseudo-random signed long integer generator

SYNOPSIS
#include <stdlib.h>

void lcong48(unsigned short int param[7]);

DESCRIPTION
Refer to drand48().

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:
• The <stdlib.h> header is now included in the SYNOPSIS section.
NAME
 ldexp — load exponent of a floating point number

SYNOPSIS
 #include <math.h>
 double ldexp(double x, int exp);

DESCRIPTION
 The ldexp() function computes the quantity $x \times 2^\text{exp}$.

RETURN VALUE
 Upon successful completion, ldexp() returns a double representing the value $x$ multiplied by 2 raised to the power $\text{exp}$.

EX
 If the value of $x$ is NaN, NaN is returned and errno may be set to [EDOM].

If ldexp() would cause overflow, ±HUGE_VAL is returned (according to the sign of $x$), and errno is set to [ERANGE].

If ldexp() would cause underflow, 0 is returned and errno may be set to [ERANGE].

ERRORS
 The ldexp() function will fail if:

[ERANGE] The value to be returned would have caused overflow.

The ldexp() function may fail if:

EX
 [EDOM] The argument $x$ is NaN.

[ERANGE] The value to be returned would have caused underflow.

No other errors will occur.

APPLICATION USAGE
 An application wishing to check for error situations should set errno to 0 before calling ldexp(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

SEE ALSO
 frexp(), isnan(), <math.h>.

CHANGE HISTORY
 First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
 The following changes are incorporated in this issue:

- Removed references to matherr().
- The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the ISO C standard and to rationalise error handling in the mathematics functions.
- The return value specified for [EDOM] is marked as an extension.
ldiv — compute quotient and remainder of a long division

The `ldiv()` function computes the quotient and remainder of the division of the numerator `numer` by the denominator `denom`. If the division is inexact, the resulting quotient is the long integer of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be represented, the behaviour is undefined; otherwise, `quot * denom + rem` will equal `numer`.

The `ldiv()` function returns a structure of type `ldiv_t`, comprising both the quotient and the remainder. The structure includes the following members, in any order:

```c
long int quot; /* quotient */
long int rem;  /* remainder */
```

No errors are defined.

```
div(), <stdlib.h>.
```

First released in Issue 4.

Derived from the ISO C standard.
NAME
lfind — find entry in linear search table

SYNOPSIS
#include <search.h>

void *lfind(const void *key, const void *base, size_t *nelp,
            size_t width, int (*compar)(const void *, const void *));

DESCRIPTION
Refer to lsearch().

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:
• In the SYNOPSIS section, the type of the function return value is changed from char* to void*, the type of the key and base arguments is changed from void* to const void*, and argument declarations for compar() are added.
NAME

lgamma — log gamma function

SYNOPSIS

EX

```c
#include <math.h>

double lgamma(double x);

extern int signgam;
```

DESCRIPTION

The `lgamma()` function computes \( \log_\Gamma(x) \) where \( \Gamma(x) \) is defined as
\[
\int_0^\infty e^{-t} t^{x-1} dt.
\]
The sign of \( \Gamma(x) \) is returned in the external integer `signgam`. The argument \( x \) may not be a non-positive integer (\( \Gamma(x) \) is defined over the reals, except the non-positive integers).

RETURN VALUE

Upon successful completion, `lgamma()` returns the logarithmic gamma of \( x \).

If \( x \) is NaN, NaN is returned and `errno` may be set to `[EDOM]`.

If \( x \) is a non-positive integer, either HUGE_VAL or NaN is returned and `errno` may be set to `[EDOM]`.

If the correct value would cause overflow, `lgamma()` returns HUGE_VAL and may set `errno` to `[ERANGE]`.

If the correct value would cause underflow, `lgamma()` returns 0 and may set `errno` to `[ERANGE]`.

ERRORS

The `lgamma()` function may fail if:

- `[EDOM]` The value of \( x \) is a non-positive integer or NaN.
- `[ERANGE]` The value to be returned would have caused overflow or underflow.

No other errors will occur.

APPLICATION USAGE

An application wishing to check for error situations should set `errno` to 0 before calling `lgamma()`. If `errno` is non-zero on return, or the return value is NaN, an error has occurred.

SEE ALSO

`exp()`, `isnan()`, `<math.h>`.

CHANGE HISTORY

First released in Issue 3.

Issue 4

The following changes are incorporated in this issue:

- This page no longer points to `gamma()`, but contains all information relating to `lgamma()`.
- The `RETURN VALUE` and `ERRORS` sections are substantially rewritten to rationalise error handling in the mathematics functions.
NAME
link — link to a file

SYNOPSIS
#include <unistd.h>

int link(const char * path1, const char * path2);

DESCRIPTION
The link() function creates a new link (directory entry) for the existing file, path1.

The path1 argument points to a pathname naming an existing file. The path2 argument points to a pathname naming the new directory entry to be created. The link() function will atomically create a new link for the existing file and the link count of the file is incremented by one.

If path1 names a directory, link() will fail unless the process has appropriate privileges and the implementation supports using link() on directories.

Upon successful completion, link() will mark for update the st_ctime field of the file. Also, the st_ctime and st_mtime fields of the directory that contains the new entry are marked for update.

If link() fails, no link is created and the link count of the file will remain unchanged.

The implementation may require that the calling process has permission to access the existing file.

RETURN VALUE
Upon successful completion, 0 is returned. Otherwise, -1 is returned and errno is set to indicate the error.

ERRORS
The link() function will fail if:

[EACCES] A component of either path prefix denies search permission, or the requested link requires writing in a directory with a mode that denies write permission, or the calling process does not have permission to access the existing file and this is required by the implementation.

[EEXIST] The link named by path2 exists.

[ELOOP] Too many symbolic links were encountered in resolving path1 or path2.

[EMLINK] The number of links to the file named by path1 would exceed {LINK_MAX}.

[ENAMETOOLONG] The length of path1 or path2 exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of either path prefix does not exist; the file named by path1 does not exist; or path1 or path2 points to an empty string.

[ENOSPC] The directory to contain the link cannot be extended.

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

[EPERM] The file named by path1 is a directory and either the calling process does not have appropriate privileges or the implementation prohibits using link() on directories.

[EROFS] The requested link requires writing in a directory on a read-only file system.
The link named by `path2` and the file named by `path1` are on different file systems and the implementation does not support links between file systems, or `path1` refers to a named STREAM.

The `link()` function may fail if:

- [ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds `{PATH_MAX}`.

**APPLICATION USAGE**

Some implementations do allow links between file systems.

**SEE ALSO**

`symlink()`, `unlink()`, `<unistd.h>`.

**CHANGE HISTORY**

First released in Issue 1.

Derived from Issue 1 of the SVID.

**Issue 4**

The following change is incorporated for alignment with the ISO POSIX-1 standard:

- The type of arguments `path1` and `path2` are changed from `char *` to `const char *`.

The following change is incorporated for alignment with the FIPS requirements:

- In the `ERRORS` section, the condition whereby [ENAMETOOLONG] will be returned if a pathname component is larger that `{NAME_MAX}` is now defined as mandatory and marked as an extension.

Other changes are incorporated as follows:

- The header `<unistd.h>` is added to the `SYNOPSIS` section.

**Issue 4, Version 2**

The `ERRORS` section is updated for X/OPEN UNIX conformance as follows:

- The [ELOOP] error will be returned if too many symbolic links are encountered during pathname resolution.

- The [EXDEV] error may also be returned if `path1` refers to a named STREAM.

- A second [ENAMETOOLONG] condition is defined that may report excessive length of an intermediate result of pathname resolution of a symbolic link.
NAME
loc1, loc2 — pointers to characters matched by regular expressions (TO BE WITHDRAWN)

SYNOPSIS

```c
#include <regexp.h>

extern char *loc1;
extern char *loc2;
```

DESCRIPTION
Refer to `regexp()`.

APPLICATION USAGE
These variables are kept for historical reasons, but will be withdrawn in a future issue of this document.

New applications should use `fnmatch()`, `glob()`, `regcomp()` and `regexec()`, which provide full internationalised regular expression functionality compatible with the ISO POSIX-2 standard, as described in the XBD specification, Chapter 7, Regular Expressions.

CHANGE HISTORY
First released in Issue 2.

Derived from Issue 2 of the SVID.

Issue 4
The following changes are incorporated in this issue:
- The header `<regexp.h>` is added to the SYNOPSIS section.
- The interface is marked TO BE WITHDRAWN, because improved functionality is now provided by interfaces introduced for alignment with the ISO POSIX-2 standard.
localeconv() — determine program locale

SYNOPSIS
#include <locale.h>
struct lconv *localeconv(void);

DESCRIPTION
The `localeconv()` function sets the components of an object with the type `struct lconv` with the
values appropriate for the formatting of numeric quantities (monetary and otherwise) according
to the rules of the current locale.

The members of the structure with type `char *` are pointers to strings, any of which (except
decimal_point) can point to "", to indicate that the value is not available in the current locale or
is of zero length. The members with type `char` are non-negative numbers, any of which can be
{CHAR_MAX} to indicate that the value is not available in the current locale.

The members include the following:

char *decimal_point
The radix character used to format non-monetary quantities.

char *thousands_sep
The character used to separate groups of digits before the decimal-point character in
formatted non-monetary quantities.

char *grouping
A string whose elements taken as one-byte integer values indicate the size of each group of
digits in formatted non-monetary quantities.

char *int_curr_symbol
The international currency symbol applicable to the current locale. The first three characters
contain the alphabetic international currency symbol in accordance with those specified in
the ISO 4217: 1987 standard. The fourth character (immediately preceding the null byte) is
the character used to separate the international currency symbol from the monetary
quantity.

char *currency_symbol
The local currency symbol applicable to the current locale.

char *mon_decimal_point
The radix character used to format monetary quantities.

char *mon_thousands_sep
The separator for groups of digits before the decimal-point in formatted monetary
quantities.

char *mon_grouping
A string whose elements taken as one-byte integer values indicate the size of each group of
digits in formatted monetary quantities.

char *positive_sign
The string used to indicate a non-negative valued formatted monetary quantity.

char *negative_sign
The string used to indicate a negative valued formatted monetary quantity.

char int_frac_digits
The number of fractional digits (those after the decimal-point) to be displayed in an
The elements of \texttt{grouping} and \texttt{mon_grouping} are interpreted according to the following:

- \texttt{[CHAR_MAX]}: No further grouping is to be performed.
- \texttt{0}: The previous element is to be repeatedly used for the remainder of the digits.
- \texttt{other}: The integer value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits before the current group.

The values of \texttt{p_sign_posn} and \texttt{n_sign_posn} are interpreted according to the following:

- \texttt{EX 0}: Parentheses surround the quantity and \texttt{currency_symbol or int_curr_symbol}.
- \texttt{EX 1}: The sign string precedes the quantity and \texttt{currency_symbol or int_curr_symbol}.
- \texttt{EX 2}: The sign string succeeds the quantity and \texttt{currency_symbol or int_curr_symbol}.
- \texttt{EX 3}: The sign string immediately precedes the \texttt{currency_symbol or int_curr_symbol}.
- \texttt{EX 4}: The sign string immediately succeeds the \texttt{currency_symbol or int_curr_symbol}.

The implementation will behave as if no function calls \texttt{localeconv()}.

\textbf{return value}

The \texttt{localeconv()} function returns a pointer to the filled-in object. The structure pointed to by the return value must not be modified by the program, but may be overwritten by a subsequent call to \texttt{localeconv()}. In addition, calls to \texttt{setlocale()} with the categories LC_ALL, LC_MONETARY, or LC_NUMERIC may overwrite the contents of the structure.
APPLICATION USAGE

The following table illustrates the rules which may be used by four countries to format monetary quantities.

<table>
<thead>
<tr>
<th>Country</th>
<th>Positive format</th>
<th>Negative format</th>
<th>International format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Italy</td>
<td>L.1.230</td>
<td>–L.1.230</td>
<td>ITL.1.230</td>
</tr>
<tr>
<td>Netherlands</td>
<td>F 1.234,56</td>
<td>F –1.234,56</td>
<td>NLG 1.234,56</td>
</tr>
<tr>
<td>Norway</td>
<td>kr1.234,56</td>
<td>kr1.234,56–</td>
<td>NOK 1.234,56</td>
</tr>
<tr>
<td>Switzerland</td>
<td>SFrs.1,234,56</td>
<td>SFrs.1,234,56C</td>
<td>CHF 1,234,56</td>
</tr>
</tbody>
</table>

For these four countries, the respective values for the monetary members of the structure returned by `localeconv()` are:

<table>
<thead>
<tr>
<th></th>
<th>Italy</th>
<th>Netherlands</th>
<th>Norway</th>
<th>Switzerland</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int_curr_symbol</code></td>
<td>&quot;ITL.&quot;</td>
<td>&quot;NLG&quot;</td>
<td>&quot;NOK&quot;</td>
<td>&quot;CHF&quot;</td>
</tr>
<tr>
<td><code>currency_symbol</code></td>
<td>&quot;L.&quot;</td>
<td>&quot;F&quot;</td>
<td>&quot;kr&quot;</td>
<td>&quot;SFrs.&quot;</td>
</tr>
<tr>
<td><code>mon_decimal_point</code></td>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td><code>mon_thousands_sep</code></td>
<td>&quot;.&quot;</td>
<td>&quot;.&quot;</td>
<td>&quot;.&quot;</td>
<td>&quot;,&quot;</td>
</tr>
<tr>
<td><code>mon_grouping</code></td>
<td>&quot;\3&quot;</td>
<td>&quot;\3&quot;</td>
<td>&quot;\3&quot;</td>
<td>&quot;\3&quot;</td>
</tr>
<tr>
<td><code>positive_sign</code></td>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td><code>negative_sign</code></td>
<td>&quot;-&quot;</td>
<td>&quot;-&quot;</td>
<td>&quot;-&quot;</td>
<td>&quot;C&quot;</td>
</tr>
<tr>
<td><code>int_frac_digits</code></td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><code>frac_digits</code></td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><code>p_cs_precedes</code></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><code>p_sep_by_space</code></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><code>n_cs_precedes</code></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><code>n_sep_by_space</code></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><code>p_sign_posn</code></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><code>n_sign_posn</code></td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

ERRORS

No errors are defined.

SEE ALSO

`isalpha()`, `isascii()`, `nl_langinfo()`, `printf()`, `scanf()`, `setlocale()`, `strcat()`, `strchr()`, `strcmp()`, `strcoll()`, `strcpy()`, `strftime()`, `strlen()`, `strpbrk()`, `strspn()`, `strtok()`, `strxfrm()`, `strtod()`, `<langinfo.h>`, `<locale.h>`.

CHANGE HISTORY

First released in Issue 4.

Derived from the ANSI C standard.
NAME
localtime — convert time value to broken-down local time

SYNOPSIS
#include <time.h>
struct tm *localtime(const time_t *timer);

DESCRIPTION
The localtime() function converts the time in seconds since the Epoch pointed to by timer into a broken-down time, expressed as a local time. The function corrects for the timezone and any seasonal time adjustments. Local timezone information is used as though localtime() calls tzset().

RETURN VALUE
The localtime() function returns a pointer to the broken-down time structure.

ERRORS
No errors are defined.

APPLICATION USAGE
Theasctime(), ctime(), getdate(), gettimeofday(), gmtime() and localtime() functions return values in one of two static objects: a broken-down time structure and an array of char. Execution of any of the functions may overwrite the information returned in either of these objects by any of the other functions.

SEE ALSO
asctime(), clock(), ctime(), difftime(), getdate(), gettimeofday(), gmtime(), mktime(), strftime(), strptime(), time(), utime(), <time.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
• The timer argument is now a type const time_t.

Another change is incorporated as follows:
• The APPLICATION USAGE section is expanded to provide a more complete description of how static areas are used by the *time() functions.
NAME
lockf — record locking on files

SYNOPSIS

```c
#include <unistd.h>

int lockf(int fildes, int function, off_t size);
```

DESCRIPTION

The `lockf()` function allows sections of a file to be locked with advisory-mode locks. Calls to `lockf()` from other processes which attempt to lock the locked file section will either return an error value or block until the section becomes unlocked. All the locks for a process are removed when the process terminates. Record locking with `lockf()` is supported for regular files and may be supported for other files.

The `fildes` argument is an open file descriptor. The file descriptor must have been opened with write-only permission (O_WRONLY) or with read/write permission (O_RDWR) to establish a lock with this function.

The `function` argument is a control value which specifies the action to be taken. The permissible values for `function` are defined in `<unistd.h>` as follows:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_ULOCK</td>
<td>unlock locked sections</td>
</tr>
<tr>
<td>F_LOCK</td>
<td>lock a section for exclusive use</td>
</tr>
<tr>
<td>F_TLOCK</td>
<td>test and lock a section for exclusive use</td>
</tr>
<tr>
<td>F_TEST</td>
<td>test a section for locks by other processes</td>
</tr>
</tbody>
</table>

`F_TEST` detects if a lock by another process is present on the specified section; `F_LOCK` and `F_TLOCK` both lock a section of a file if the section is available; `F_ULOCK` removes locks from a section of the file.

The `size` argument is the number of contiguous bytes to be locked or unlocked. The section to be locked or unlocked starts at the current offset in the file and extends forward for a positive size or backward for a negative size (the preceding bytes up to but not including the current offset). If `size` is 0, the section from the current offset through the largest possible file offset is locked (that is, from the current offset through the present or any future end-of-file). An area need not be allocated to the file to be locked because locks may exist past the end-of-file.

The sections locked with `F_LOCK` or `F_TLOCK` may, in whole or in part, contain or be contained by a previously locked section for the same process. When this occurs, or if adjacent locked sections would occur, the sections are combined into a single locked section. If the request would cause the number of locks to exceed a system-imposed limit, the request will fail.

`F_LOCK` and `F_TLOCK` requests differ only by the action taken if the section is not available. `F_LOCK` blocks the calling process until the section is available. `F_TLOCK` makes the function fail if the section is already locked by another process.

File locks are released on first close by the locking process of any file descriptor for the file.

`F_ULOCK` requests may release (wholly or in part) one or more locked sections controlled by the process. Locked sections will be unlocked starting at the current file offset through `size` bytes or to the end of file if `size` is (off_t)0. When all of a locked section is not released (that is, when the beginning or end of the area to be unlocked falls within a locked section), the remaining portions of that section are still locked by the process. Releasing the center portion of a locked section will cause the remaining locked beginning and end portions to become two separate locked sections. If the request would cause the number of locks in the system to exceed a system-
imposed limit, the request will fail.

A potential for deadlock occurs if a process controlling a locked section is blocked by accessing another process' locked section. If the system detects that deadlock would occur, lockf() will fail with an [EDEADLK] error.

The interaction between fcntl() and lockf() locks is unspecified.

Blocking on a section is interrupted by any signal.

RETURN VALUE

Upon successful completion, lockf() returns 0. Otherwise, it returns −1, sets errno to indicate an error, and existing locks are not changed.

ERRORS

The lockf() function will fail if:

- [EBADF] The fildes argument is not a valid open file descriptor; or function is F_LOCK or F_TLOCK and fildes is not a valid file descriptor open for writing.
- [EACCES] or [EAGAIN] The function argument is F_TLOCK or F_TEST and the section is already locked by another process.
- [EDEADLK] The function argument is F_LOCK and a deadlock is detected.
- [EINTR] A signal was caught during execution of the function.

The lockf() function may fail if:

- [EAGAIN] The function argument is F_LOCK or F_TLOCK and the file is mapped with mmap().
- [EDEADLK] or [ENOLCK] The function argument is F_LOCK, F_TLOCK, or F_ULOCK, and the request would cause the number of locks to exceed a system-imposed limit.
- [EOPNOTSUPP] or [EINVAL] The implementation does not support the locking of files of the type indicated by the fildes argument.
- [EINVAL] The function argument is not one of F_LOCK, F_TLOCK, F_TEST or F_ULOCK; or size plus the current file offset is less than 0 or greater than the largest possible file offset.

APPLICATION USAGE

Record-locking should not be used in combination with the fopen(), fread(), fwrite() and other stdio functions. Instead, the more primitive, non-buffered functions (such as open()) should be used. Unexpected results may occur in processes that do buffering in the user address space. The process may later read/write data which is/was locked. The stdio functions are the most common source of unexpected buffering.

The alarm() function may be used to provide a timeout facility in applications requiring it.

SEE ALSO

alarm(), chmod(), close(), creat(), fcntl(), mmap(), open(), read(), write(), <unistd.h>.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
locs — stop regular expression matching in a string (TO BE WITHDRAWN)

SYNOPSIS

```c
#include <regexp.h>

extern char *locs;
```

DESCRIPTION
Refer to `regexp()`.

APPLICATION USAGE
This variable is kept for historical reasons, but will be withdrawn in a future issue of this document.

New applications should use `fnmatch()`, `glob()`, `regcomp()` and `regexec()`, which provide full internationalised regular expression functionality compatible with the ISO POSIX-2 standard, as described in the XBD specification, Chapter 7, Regular Expressions.

CHANGE HISTORY
First released in Issue 2.
Derived from Issue 2 of the SVID.

Issue 4
The following changes are incorporated in this issue:

- The header `<regexp.h>` is added to the SYNOPSIS section.
- The interface is marked TO BE WITHDRAWN, because improved functionality is now provided by interfaces introduced for alignment with the ISO POSIX-2 standard.
NAME
log — natural logarithm function

SYNOPSIS
#include <math.h>
double log(double x);

DESCRIPTION
The log() function computes the natural logarithm of x, \( \log_e(x) \). The value of x must be positive.

RETURN VALUE
Upon successful completion, log() returns the natural logarithm of x.

EX
If x is NaN, NaN is returned and errno may be set to [EDOM].

EX
If x is less than 0, –HUGE_VAL or NaN is returned, and errno is set to [EDOM].
If x is 0, –HUGE_VAL is returned and errno may be set to [ERANGE].

ERRORS
The log() function will fail if:
[EDOM] The value of x is negative.

The log() function may fail if:
[EDOM] The value of x is NaN.

[ERANGE] The value of x is 0.

EX
No other errors will occur.

APPLICATION USAGE
An application wishing to check for error situations should set errno to 0 before calling log(). If
errno is non-zero on return, or the return value is NaN, an error has occurred.

SEE ALSO
exp(), isnan(), log10(), log1p(), <math.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:
• Removed references to matherr().
• The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with
  the ISO C standard and to rationalise error handling in the mathematics functions.
• The return value specified for [EDOM] is marked as an extension.
NAME
log10 — base 10 logarithm function

SYNOPSIS
#include <math.h>

double log10(double x);

DESCRIPTION
The log10() function computes the base 10 logarithm of x, \log_{10}(x). The value of x must be positive.

RETURN VALUE
Upon successful completion, log10() returns the base 10 logarithm of x.

EX
Ex If x is NaN, NaN is returned and errno may be set to [EDOM].
Ex If x is less than 0, −HUGE_VAL or NaN is returned, and errno is set to [EDOM].
If x is 0, −HUGE_VAL is returned and errno may be set to [ERANGE].

ERRORS
The log10() function will fail if:
[EDOM] The value of x is negative.

The log10() function may fail if:
[EDOM] The value of x is NaN.
[ERANGE] The value of x is 0.

EX No other errors will occur.

APPLICATION USAGE
An application wishing to check for error situations should set errno to 0 before calling log10(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

SEE ALSO
isnan(), log(), pow(), <math.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:
• Removed references to matherr().
• The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the ISO C standard and to rationalise error handling in the mathematics functions.
• The return value specified for [EDOM] is marked as an extension.
NAME
log1p — compute natural logarithm

SYNOPSIS
ux
#include <math.h>

double log1p (double x);

DESCRIPTION
The log1p() function computes \( \log_e(1.0 + x) \). The value of \( x \) must be greater than \(-1.0\).

RETURN VALUE
Upon successful completion, log1p() returns the natural logarithm of \( 1.0 + x \).
If \( x \) is NaN, log1p() returns NaN and may set errno to [EDOM].
If \( x \) is less than \(-1.0\), log1p() returns \(-\text{HUGE}_\text{VAL}\) or NaN and sets errno to [EDOM].
If \( x \) is \(-1.0\), log1p() returns \(-\text{HUGE}_\text{VAL}\) and may set errno to [ERANGE].

ERRORS
The log1p() function will fail if:
[EDOM] The value of \( x \) is less than \(-1.0\).
The log1p() function may fail and set errno to:
[EDOM] The value of \( x \) is NaN.
[ERANGE] The value of \( x \) is \(-1.0\).
No other errors will occur.

SEE ALSO
log(), <math.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
logb — radix-independent exponent

SYNOPSIS
#include <math.h>

double logb(double x);

DESCRIPTION
The logb() function computes the exponent of x, which is the integral part of \( \log_r |x| \), as a signed floating point value, for non-zero x, where r is the radix of the machine’s floating-point arithmetic.

RETURN VALUE
Upon successful completion, logb() returns the exponent of x.
If x is 0.0, logb() returns -HUGE_VAL and sets errno to [EDOM].
If x is ±Inf, logb() returns +Inf.
If x is NaN, logb() returns NaN and may set errno to [EDOM].

ERRORS
The logb() function will fail if:
[EDOM] The x argument is 0.0.
The logb() function may fail if:
[EDOM] The x argument is NaN.

SEE ALSO
ilogb(), <math.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
_longjmp, _setjmp — non-local goto

SYNOPSIS
UX
#include <setjmp.h>

void _longjmp(jmp_buf env, int val);

int _setjmp(jmp_buf env);

DESCRIPTION
The _longjmp() and _setjmp() functions are identical to longjmp() and setjmp(), respectively, with
the additional restriction that _longjmp() and _setjmp() do not manipulate the signal mask.

If _longjmp() is called even though env was never initialised by a call to _setjmp(), or when the
last such call was in a function that has since returned, the results are undefined.

RETURN VALUE
Refer to longjmp() and setjmp().

ERRORS
No errors are defined.

APPLICATION USAGE
If _longjmp() is executed and the environment in which _setjmp() was executed no longer exists,
errors can occur. The conditions under which the environment of the _setjmp() no longer exists
include exiting the function that contains the _setjmp() call, and exiting an inner block with
temporary storage. This condition might not be detectable, in which case the _longjmp() occurs
and, if the environment no longer exists, the contents of the temporary storage of an inner block
are unpredictable. This condition might also cause unexpected process termination. If the
function has returned, the results are undefined.

Passing longjmp() a pointer to a buffer not created by setjmp(), passing _longjmp() a pointer to a
buffer not created by _setjmp(), passing siglongjmp() a pointer to a buffer not created by
sigsetjmp() or passing any of these three functions a buffer that has been modified by the user
can cause all the problems listed above, and more.

The _longjmp() and _setjmp() functions are included to support programs written to historical
system interfaces. New applications should use siglongjmp() and sigsetjmp() respectively.

SEE ALSO
longjmp(), setjmp(), siglongjmp(), sigsetjmp(), <setjmp.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
longjmp — non-local goto

SYNOPSIS
#include <setjmp.h>

void longjmp(jmp_buf env, int val);

DESCRIPTION
The longjmp() function restores the environment saved by the most recent invocation of setjmp() in the same process, with the corresponding jmp_buf argument. If there is no such invocation, or if the function containing the invocation of setjmp() has terminated execution in the interim, the behaviour is undefined. It is unspecified whether longjmp() restores the signal mask, leaves the signal mask unchanged or restores it to its value at the time setjmp() was called.

All accessible objects have values as of the time longjmp() was called, except that the values of objects of automatic storage duration are indeterminate if they meet all the following conditions:

• They are local to the function containing the corresponding setjmp() invocation.
• They do not have volatile-qualified type.
• They are changed between the setjmp() invocation and longjmp() call.

As it bypasses the usual function call and return mechanisms, longjmp() will execute correctly in contexts of interrupts, signals and any of their associated functions. However, if longjmp() is invoked from a nested signal handler (that is, from a function invoked as a result of a signal raised during the handling of another signal), the behaviour is undefined.

RETURN VALUE
After longjmp() is completed, program execution continues as if the corresponding invocation of setjmp() had just returned the value specified by val. The longjmp() function cannot cause setjmp() to return 0; if val is 0, setjmp() returns 1.

ERRORS
No errors are defined.

APPLICATION USAGE
Applications whose behaviour depends on the value of the signal mask should not use longjmp() and setjmp(), since their effect on the signal mask is unspecified, but should instead use the following alternatives:
• The _longjmp() and _setjmp() functions (which never modify the signal mask)
• The siglongjmp() and sigsetjmp() functions (which can save and restore the signal mask under application control).

SEE ALSO
setjmp(), sigaction(), siglongjmp(), sigsetjmp(), <setjmp.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
• Mention of volatile-qualified types is added to the DESCRIPTION section.
Another change is incorporated as follows:

- The APPLICATION USAGE section is deleted.

**Issue 4, Version 2**

The DESCRIPTION is updated for X/OPEN UNIX conformance and discusses valid possibilities for the resulting state of the signal mask.
NAME
lrand48 — generate uniformly distributed pseudo-random non-negative long integers

SYNOPSIS
EX
#include <stdlib.h>
long int lrand48(void);

DESCRIPTION
Refer to drand48().

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:
• The <stdlib.h> header is now included in the SYNOPSIS section.
• The argument list now contains void.
NAME
lsearch, lfind — linear search and update

SYNOPSIS

EX
#include <search.h>

void *lsearch(const void * key , void * base , size_t * nelp , size_t width ,
              int (* compar )(const void *, const void *));

void *lfind(const void * key , const void * base , size_t * nelp ,
            size_t width, int (*compar)(const void *, const void *));

DESCRIPTION
The lsearch() function is a linear search routine. It returns a pointer into a table indicating where
an entry may be found. If the entry does not occur, it is added at the end of the table. The key
argument points to the entry to be sought in the table. The base argument points to the first
element in the table. The width argument is the size of an element in bytes. The nelp argument
points to an integer containing the current number of elements in the table. The integer to which
nelp points is incremented if the entry is added to the table. The compar argument points to a
comparison function which the user must supply (strcmp(), for example). It is called with two
arguments that point to the elements being compared. The function must return 0 if the
elements are equal and non-zero otherwise.

The lfind() function is the same as lsearch() except that if the entry is not found, it is not added to
the table. Instead, a null pointer is returned.

RETURN VALUE
If the searched for entry is found, both lsearch() and lfind() return a pointer to it. Otherwise,
lfind() returns a null pointer and lsearch() returns a pointer to the newly added element.

Both functions return a null pointer in case of error.

ERRORS
No errors are defined.

EXAMPLES
This fragment will read in less than or equal to TABSIZE strings of length less than or equal to
ELSIZE and store them in a table, eliminating duplicates.

#include <stdio.h>
#include <string.h>
#include <search.h>
#define TABSIZE 50
#define ELSIZE 120
...
char line[ELSIZE], tab[TABSIZE][ELSIZE];
size_t nel = 0;
...,
while (fgets(line, ELSIZE, stdin) != NULL & & nel < TABSIZE)
(void) lsearch(line, tab, &nel,
              ELSIZE, (int *)(const void *, const void *) strcmp);
...
APPLICATION USAGE
The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

Undefined results can occur if there is not enough room in the table to add a new item.

SEE ALSO
bsearch(), hsearch(), tsearch(), <search.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

• In the SYNOPSIS section, the type of argument key in the declaration of lsearch() is changed from void* to const void*, the type arguments key and base have been changed from void* to const void* in the declaration of lfind(), and the arguments to compar() are defined for both functions.

• In the EXAMPLES section, the sample code is updated to use ISO C syntax.

• Warnings about the casting of various arguments are removed from the APPLICATION USAGE section, as casting requirements are now clear from the function definitions.
NAME
lseek — move read/write file offset

SYNOPSIS

```c
#include <sys/types.h>
#include <unistd.h>

off_t lseek(int fildes, off_t offset, int whence);
```

DESCRIPTION
The `lseek()` function will set the file offset for the open file description associated with the file
descriptor `fildes`, as follows:

- If `whence` is SEEK_SET the file offset is set to `offset` bytes.
- If `whence` is SEEK_CUR the file offset is set to its current location plus `offset`.
- If `whence` is SEEK_END the file offset is set to the size of the file plus `offset`.

The symbolic constants SEEK_SET, SEEK_CUR and SEEK_END are defined in the header
<unistd.h>.

The behaviour of `lseek()` on devices which are incapable of seeking is implementation-
dependent. The value of the file offset associated with such a device is undefined.

The `lseek()` function will allow the file offset to be set beyond the end of the existing data in the
file. If data is later written at this point, subsequent reads of data in the gap will return bytes
with the value 0 until data is actually written into the gap.

The `lseek()` function will not, by itself, extend the size of a file.

RETURN VALUE
Upon successful completion, the resulting offset, as measured in bytes from the beginning of the
file, is returned. Otherwise, `(off_t)-1` is returned, `errno` is set to indicate the error and the file
offset will remain unchanged.

ERRORS
The `lseek()` function will fail if:

- [EBADF] The `fildes` argument is not an open file descriptor.
- [EINVAL] The `whence` argument is not a proper value, or the resulting file offset would be
  invalid.
- [ESPIPE] The `fildes` argument is associated with a pipe or FIFO.

SEE ALSO
`open()`, <sys/tyes.h>, <unistd.h>.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

- The header <sys/types.h> is now marked as optional (OH); this header need not be included
  on XSI-conformant systems.
- The APPLICATION USAGE section is removed, as the ISO POSIX-1 standard now requires
  that `off_t` be signed.
NAME
lstat — get symbolic link status

SYNOPSIS

\#include <sys/stat.h>

int lstat(const char *path, struct stat *buf);

DESCRIPTION

The lstat() function has the same effect as stat(), except when path refers to a symbolic link. In that case lstat() returns information about the link, while stat() returns information about the file the link references.

For symbolic links, the st_mode member will contain meaningful information when used with the file type macros, and the st_size member will contain the length of the pathname contained in the symbolic link. File mode bits and the contents of the remaining members of the stat structure are unspecified. The value returned in the st_size member is the length of the contents of the symbolic link, and does not count any trailing null.

RETURN VALUE

Upon successful completion, lstat() returns 0. Otherwise, it returns -1 and sets errno to indicate the error.

ERRORS

The lstat() function will fail if:

- [EACCES] A component of the path prefix denies search permission.
- [EIO] An error occurred while reading from the file system.
- [ELOOP] Too many symbolic links were encountered in resolving path.
- [ENAMETOOLONG] The length of a pathname exceeds [PATH_MAX], or pathname component is longer than [NAME_MAX].
- [ENOTDIR] A component of the path prefix is not a directory.
- [ENOENT] A component of path does not name an existing file or path is an empty string.

The lstat() function may fail if:

- [ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds [PATH_MAX].
- [EOVERFLOW] One of the members is too large to store into the structure pointed to by the buf argument.

SEE ALSO

fstat(), readlink(), stat(), symlink(), <sys/stat.h>.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
makecontext, swapcontext — manipulate user contexts

SYNOPSIS
```
#include <ucontext.h>

void makecontext(ucontext_t * ucp, (void *func)(), int argc, ...);
int swapcontext(ucontext_t * oucp, const ucontext_t * ucp);
```

DESCRIPTION
The makecontext() function modifies the context specified by ucp, which has been initialised using getcontext(). When this context is resumed using swapcontext() or setcontext(), program execution continues by calling func(), passing it the arguments that follow argc in the makecontext() call.

Before a call is made to makecontext(), the context being modified should have a stack allocated for it. The value of argc must match the number of integer arguments passed to func(), otherwise the behaviour is undefined.

The uc_link member is used to determine the context that will be resumed when the context being modified by makecontext() returns. The uc_link member should be initialised prior to the call to makecontext().

The swapcontext() function saves the current context in the context structure pointed to by oucp and sets the context to the context structure pointed to by ucp.

RETURN VALUE
On successful completion, swapcontext() returns 0. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS
The makecontext() and swapcontext() functions will fail if:

[ENOMEM] The ucp argument does not have enough stack left to complete the operation.

SEE ALSO
exit(), getcontext(), sigaction(), sigprocmask(), <ucontext.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
malloc — memory allocator

SYNOPSIS
#include <stdlib.h>
void *malloc(size_t size);

DESCRIPTION
The malloc() function allocates unused space for an object whose size in bytes is specified by size and whose value is indeterminate.

The order and contiguity of storage allocated by successive calls to malloc() is unspecified. The pointer returned if the allocation succeeds is suitably aligned so that it may be assigned to a pointer to any type of object and then used to access such an object in the space allocated (until the space is explicitly freed or reallocated). Each such allocation will yield a pointer to an object disjoint from any other object. The pointer returned points to the start (lowest byte address) of the allocated space. If the space cannot be allocated, a null pointer is returned. If the size of the space requested is 0, the behaviour is implementation-dependent; the value returned will be either a null pointer or a unique pointer.

RETURN VALUE
Upon successful completion with size not equal to 0, malloc() returns a pointer to the allocated space. If size is 0, either a null pointer or a unique pointer that can be successfully passed to free() will be returned. Otherwise, it returns a null pointer and sets errno to indicate the error.

ERRORS
The malloc() function will fail if:

[ENOMEM] Insufficient storage space is available.

APPLICATION USAGE
There is now no requirement for the implementation to support the inclusion of <malloc.h>.

SEE ALSO
calloc(), free(), realloc(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
• The RETURN VALUE section is updated to indicate what will be returned if size is 0.

Other changes are incorporated as follows:
• The setting of errno and the [ENOMEM] error are marked as extensions.
• The APPLICATION USAGE section is changed to record that <malloc.h> need no longer be supported on XSI-conformant systems.
NAME
mblen — get number of bytes in a character

SYNOPSIS
#include <stdlib.h>
int mblen(const char *s, size_t n);

DESCRIPTION
If s is not a null pointer, mblen() determines the number of bytes constituting the character
pointed to by s. Except that the shift state of mbtowc() is not affected, it is equivalent to:

mbtowc((wchar_t *)0, s, n);

The implementation will behave as if no function defined in this document calls mblen().

The behaviour of this function is affected by the LC_CTYPE category of the current locale. For a
state-dependent encoding, this function is placed into its initial state by a call for which its
character pointer argument, s, is a null pointer. Subsequent calls with s as other than a null
pointer cause the internal state of the function to be altered as necessary. A call with s as a null
pointer causes this function to return a non-zero value if encodings have state dependency, and
0 otherwise. If the implementation employs special bytes to change the shift state, these bytes
do not produce separate wide-character codes, but are grouped with an adjacent character.
Changing the LC_CTYPE category causes the shift state of this function to be indeterminate.

RETURN VALUE
If s is a null pointer, mblen() returns a non-zero or 0 value, if character encodings, respectively,
do or do not have state-dependent encodings. If s is not a null pointer, mblen() either returns 0
(if s points to the null byte), or returns the number of bytes that constitute the character (if the
next n or fewer bytes form a valid character), or returns \(-1\) (if they do not form a valid character)
and may set errno to indicate the error. In no case will the value returned be greater than n or the
value of the MB_CUR_MAX macro.

ERRORS
The mblen() function may fail if:

EX [EILSEQ] Invalid character sequence is detected.

SEE ALSO
mbtowc(), mbstowcs(), wctomb(), wcstombs(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 4.
Aligned with the ISO C standard.
NAME
mbstowcs — convert a character string to a wide character string

SYNOPSIS
#include <stdlib.h>
size_t mbstowcs(wchar_t * pwcs, const char * s, size_t n);

DESCRIPTION
The mbstowcs() function converts a sequence of characters that begins in the initial shift state from the array pointed to by s into a sequence of corresponding wide-character codes and stores not more than n wide-character codes into the array pointed to by pwcs. No characters that follow a null byte (which is converted into a wide-character code with value 0) will be examined or converted. Each character is converted as if by a call to mbtowc(), except that the shift state of mbtowc() is not affected.

No more than n elements will be modified in the array pointed to by pwcs. If copying takes place between objects that overlap, the behaviour is undefined.

EX
The behaviour of this function is affected by the LC_CTYPE category of the current locale. If pwcs is a null pointer, mbstowcs() returns the length required to convert the entire array regardless of the value of n, but no values are stored.

RETURN VALUE
If an invalid character is encountered, mbstowcs() returns (size_t)−1 and may set errno to indicate the error. Otherwise, mbstowcs() returns the number of the array elements modified (or required if pwcs is null), not including a terminating 0 code, if any. The array will not be zero-terminated if the value returned is n.

ERRORS
The mbstowcs() function may fail if:

EX
[EILSEQ] Invalid byte sequence is detected.

SEE ALSO
mbtowc(), wcstombs(), mblen(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 4.

Aligned with the ISO C standard.
NAME
mbtowc — convert a character to a wide-character code

SYNOPSIS
#include <stdlib.h>

int mbtowc(wchar_t *pwc, const char *s, size_t n);

DESCRIPTION
If *s* is not a null pointer, *mbtowc()* determines the number of the bytes that constitute the
character pointed to by *s*. It then determines the wide-character code for the value of type
wchar_t that corresponds to that character. (The value of the wide-character code
corresponding to the null byte is 0.) If the character is valid and *pwc* is not a null pointer,
*mbtowc()* stores the wide-character code in the object pointed to by *pwc*.

The behaviour of this function is affected by the LC_CTYPE category of the current locale. For a
state-dependent encoding, this function is placed into its initial state by a call for which its
character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null
pointer cause the internal state of the function to be altered as necessary. A call with *s* as a null
pointer causes this function to return a non-zero value if encodings have state dependency, and
0 otherwise. If the implementation employs special bytes to change the shift state, these bytes
do not produce separate wide-character codes, but are grouped with an adjacent character.
Changing the LC_CTYPE category causes the shift state of this function to be indeterminate. At
most *n* bytes of the array pointed to by *s* will be examined.

The implementation will behave as if no function defined in this document calls *mbtowc()*.

RETURN VALUE
If *s* is a null pointer, *mbtowc()* returns a non-zero or 0 value, if character encodings, respectively,
do or do not have state-dependent encodings. If *s* is not a null pointer, *mbtowc()* either returns 0
(if *s* points to the null byte), or returns the number of bytes that constitute the converted
character (if the next *n* or fewer bytes form a valid character), or returns −1 and may set errno to
indicate the error (if they do not form a valid character).

In no case will the value returned be greater than *n* or the value of the MB_CUR_MAX macro.

ERRORS
The *mbtowc()* function may fail if:

EX [EILSEQ] Invalid character sequence is detected.

SEE ALSO
mblen(), mbstowcs(), wctomb(), wcstombs(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 4.
Aligned with the ISO C standard.
NAME
memccpy — copy bytes in memory

SYNOPSIS
EX
#include <string.h>

void *memccpy(void *s1, const void *s2, int c, size_t n);

DESCRIPTION
The memccpy() function copies bytes from memory area s2 into s1, stopping after the first occurrence of byte c (converted to an unsigned char) is copied, or after n bytes are copied, whichever comes first. If copying takes place between objects that overlap, the behaviour is undefined.

RETURN VALUE
The memccpy() function returns a pointer to the byte after the copy of c in s1, or a null pointer if c was not found in the first n bytes of s2.

ERRORS
No errors are defined.

APPLICATION USAGE
The memccpy() function does not check for the overflow of the receiving memory area.

SEE ALSO
<string.h>.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

• The type of argument s2 is changed from void* to const void*.

• Reference to use of the header <memory.h> is removed from the APPLICATION USAGE section.

• The FUTURE DIRECTIONS section is removed.
NAME
memchr — find byte in memory

SYNOPSIS
#include <string.h>

void *memchr(const void *s, int c, size_t n);

DESCRIPTION
The memchr() function locates the first occurrence of c (converted to an unsigned char) in the initial n bytes (each interpreted as unsigned char) of the object pointed to by s.

RETURN VALUE
The memchr() function returns a pointer to the located byte, or a null pointer if the byte does not occur in the object.

ERRORS
No errors are defined.

SEE ALSO
<string.h>.

CHANGE HISTORY
First released in Issue 1.
 Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated for alignment with the ISO C standard:
• The function is no longer marked as an extension.
• The type of argument s is changed from void* to const void*.

Another change is incorporated as follows:
• The APPLICATION USAGE section is removed.
NAME
  memcmp — compare bytes in memory

SYNOPSIS
  #include <string.h>
  int memcmp(const void * s1, const void * s2, size_t n);

DESCRIPTION
  The memcmp() function compares the first n bytes (each interpreted as unsigned char) of
  the object pointed to by s1 to the first n bytes of the object pointed to by s2.

  The sign of a non-zero return value is determined by the sign of the difference between
  the values of the first pair of bytes (both interpreted as type unsigned char) that differ
  in the objects being compared.

RETURN VALUE
  The memcmp() function returns an integer greater than, equal to or less than 0, if the object
  pointed to by s1 is greater than, equal to or less than the object pointed to by s2 respectively.

ERRORS
  No errors are defined.

SEE ALSO
  <string.h>.

CHANGE HISTORY
  First released in Issue 1.
  Derived from Issue 1 of the SVID.

Issue 4
  The following changes are incorporated for alignment with the ISO C standard:
    • The function is no longer marked as an extension.
    • The type of arguments s1 and s2 are changed from void* to const void*.

Other changes are incorporated as follows:
  • The RETURN VALUE section is clarified.
  • The APPLICATION USAGE section is removed.
NAME
memcpy — copy bytes in memory

SYNOPSIS
#include <string.h>

void *memcpy(void * s1, const void * s2, size_t n);

DESCRIPTION
The memcpy() function copies n bytes from the object pointed to by s2 into the object pointed to by s1. If copying takes place between objects that overlap, the behaviour is undefined.

RETURN VALUE
The memcpy() function returns s1; no return value is reserved to indicate an error.

ERRORS
No errors are defined.

APPLICATION USAGE
The memcpy() function does not check for the overflowing of the receiving memory area.

SEE ALSO
<string.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated for alignment with the ISO C standard:
• The function is no longer marked as an extension.
• The type of argument s2 is changed from void* to const void*.

Other changes are incorporated as follows:
• Reference to use of the header <memory.h> is removed from the APPLICATION USAGE section, and a note about overflow checking has been added.
• The FUTURE DIRECTIONS section is removed.
NAME

memmove — copy bytes in memory with overlapping areas

SYNOPSIS

#include <string.h>

void *memmove(void *s1, const void *s2, size_t n);

DESCRIPTION

The memmove() function copies n bytes from the object pointed to by s2 into the object pointed to by s1. Copying takes place as if the n bytes from the object pointed to by s2 are first copied into a temporary array of n bytes that does not overlap the objects pointed to by s1 and s2, and then the n bytes from the temporary array are copied into the object pointed to by s1.

RETURN VALUE

The memmove() function returns s1; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

SEE ALSO

<string.h>.

CHANGE HISTORY

First released in Issue 4.

Derived from the ANSI C standard.
NAME
memset — set bytes in memory

SYNOPSIS
#include <string.h>
void *memset(void *s, int c, size_t n);

DESCRIPTION
The memset() function copies c (converted to an unsigned char) into each of the first n bytes of the object pointed to by s.

RETURN VALUE
The memset() function returns s; no return value is reserved to indicate an error.

ERRORS
No errors are defined.

SEE ALSO
<string.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:

• The function is no longer marked as an extension.

Another change is incorporated as follows:

• The APPLICATION USAGE section is removed.
NAME
mkdir — make a directory

SYNOPSIS
#include <sys/types.h>
#include <sys/stat.h>

int mkdir(const char *path, mode_t mode);

DESCRIPTION
The mkdir() function creates a new directory with name path. The file permission bits of the new
directory are initialised from mode. These file permission bits of the mode argument are modified
by the process’ file creation mask.

When bits in mode other than the file permission bits are set, the meaning of these additional bits
is implementation-dependent.

The directory’s user ID is set to the process’ effective user ID. The directory’s group ID is set to
the group ID of the parent directory or to the effective group ID of the process.

The newly created directory will be an empty directory.

Upon successful completion, mkdir() will mark for update the st_atime, st_ctime and st_mtime
fields of the directory. Also, the st_ctime and st_mtime fields of the directory that contains the
new entry are marked for update.

RETURN VALUE
Upon successful completion, mkdir() returns 0. Otherwise, −1 is returned, no directory is created
and errno is set to indicate the error.

ERRORS
The mkdir() function will fail if:

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

[EXDEV] The named file exists.

[ELoop] Too many symbolic links were encountered in resolving path.

[ENAMETOOLONG] The length of the path argument exceeds [PATH_MAX] or a pathname component is longer than [NAME_MAX].

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

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

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

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

The mkdir() function may fail if:

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result
whose length exceeds [PATH_MAX].
SEE ALSO

`umask()`, `<sys/stat.h>`, `<sys/types.h>`.

CHANGE HISTORY

First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

**Issue 4**

The following change is incorporated for alignment with the ISO POSIX-1 standard:

- The type of argument `path` is changed from `char *` to `const char *`.

The following changes are incorporated for alignment with the FIPS requirements:

- In the **ERRORS** section, the condition whereby `[ENAMETOOLONG]` will be returned if a pathname component is larger than `{NAME_MAX}` is now defined as mandatory and marked as an extension.

Another change is incorporated as follows:

- The header `<sys/types.h>` is now marked as optional (oh); this header need not be included on XSI-conformant systems.

**Issue 4, Version 2**

The **ERRORS** section is updated for X/OPEN UNIX conformance as follows:

- It states that `[ELOOP]` will be returned if too many symbolic links are encountered during pathname resolution.
- A second `[ENAMETOOLONG]` condition is defined that may report excessive length of an intermediate result of pathname resolution of a symbolic link.
NAME
mkfifo — make a FIFO special file

SYNOPSIS
#include <sys/types.h>
#include <sys/stat.h>

int mkfifo(const char *path, mode_t mode);

DESCRIPTION
The mkfifo() function creates a new FIFO special file named by the pathname pointed to by path. The file permission bits of the new FIFO are initialised from mode. The file permission bits of the mode argument are modified by the process' file creation mask.

When bits in mode other than the file permission bits are set, the effect is implementation-dependent.

The FIFO's user ID will be set to the process' effective user ID. The FIFO's group ID will be set to the group ID of the parent directory or to the effective group ID of the process.

Upon successful completion, mkfifo() will mark for update the st_atime, st_ctime and st_mtime fields of the file. Also, the st_ctime and st_mtime fields of the directory that contains the new entry are marked for update.

RETURN VALUE
Upon successful completion, 0 is returned. Otherwise, -1 is returned, no FIFO is created and errno is set to indicate the error.

ERRORS
The mkfifo() function will fail if:

[EACCES] A component of the path prefix denies search permission, or write permission is denied on the parent directory of the FIFO to be created.

[EEXIST] The named file already exists.

[ELoop] Too many symbolic links were encountered in resolving path.

[ENAMEETOOLONG] The length of the path argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.

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

[ENOSPC] The directory that would contain the new file cannot be extended or the file system is out of file-allocation resources.

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

[EROFS] The named file resides on a read-only file system.

The mkfifo() function may fail if:

[ENAMEETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}. 

X/Open CAE Specification (1994)
SEE ALSO

`umask()`, `<sys/stat.h>`, `<sys/types.h>`.

CHANGE HISTORY

First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

Issue 4

The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- The type of argument `path` is changed from `char *` to `const char *`.
- The description of [EACCES] is updated to indicate that this error will also be returned if write permission is denied to the parent directory.

The following changes are incorporated for alignment with the FIPS requirements:

- In the ERRORS section, the condition whereby [ENAMETOOLONG] will be returned if a pathname component is larger that [NAME_MAX] is now defined as mandatory and marked as an extension.

Another change is incorporated as follows:

- The header `<sys/types.h>` is now marked as optional (OH); this header need not be included on XSI-conformant systems.

Issue 4, Version 2

The ERRORS section is updated for X/OPEN UNIX conformance as follows:

- It states that [ELOOP] will be returned if too many symbolic links are encountered during pathname resolution.
- A second [ENAMETOOLONG] condition is defined that may report excessive length of an intermediate result of pathname resolution of a symbolic link.
NAME

mknod — make a directory, a special or regular file

SYNOPSIS

```c
#include <sys/stat.h>

int mknod(const char *path, mode_t mode, dev_t dev);
```

DESCRIPTION

The `mknod()` function creates a new file named by the pathname to which the argument `path` points.

The file type for `path` is OR-ed into the `mode` argument, and must be selected from one of the following symbolic constants:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IFIFO</td>
<td>FIFO-special</td>
</tr>
<tr>
<td>S_IFCHR</td>
<td>Character-special (non-portable)</td>
</tr>
<tr>
<td>S_IFDIR</td>
<td>Directory (non-portable)</td>
</tr>
<tr>
<td>S_IFBLK</td>
<td>Block-special (non-portable)</td>
</tr>
<tr>
<td>S_IFREG</td>
<td>Regular (non-portable)</td>
</tr>
</tbody>
</table>

The only portable use of `mknod()` is to create a FIFO-special file. If `mode` is not `S_IFIFO` or `dev` is not 0, the behaviour of `mknod()` is unspecified.

The permissions for the new file are OR-ed into the `mode` argument, and may be selected from any combination of the following symbolic constants:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_ISUID</td>
<td>Set user ID on execution.</td>
</tr>
<tr>
<td>S_ISGID</td>
<td>Set group ID on execution.</td>
</tr>
<tr>
<td>S_IRWXU</td>
<td>Read, write or execute (search) by owner.</td>
</tr>
<tr>
<td>S_IRUSR</td>
<td>Read by owner.</td>
</tr>
<tr>
<td>S_IWUSR</td>
<td>Write by owner.</td>
</tr>
<tr>
<td>S_IXUSR</td>
<td>Execute (search) by owner.</td>
</tr>
<tr>
<td>S_IRWGX</td>
<td>Read, write or execute (search) by group.</td>
</tr>
<tr>
<td>S_IRGRP</td>
<td>Read by group.</td>
</tr>
<tr>
<td>S_IWGRP</td>
<td>Write by group.</td>
</tr>
<tr>
<td>S_IXGRP</td>
<td>Execute (search) by group.</td>
</tr>
<tr>
<td>S_IRWXO</td>
<td>Read, write or execute (search) by others.</td>
</tr>
<tr>
<td>S_IROTH</td>
<td>Read by others.</td>
</tr>
<tr>
<td>S_IWOTH</td>
<td>Write by others.</td>
</tr>
<tr>
<td>S_IXOTH</td>
<td>Execute (search) by others.</td>
</tr>
<tr>
<td>S_ISVTX</td>
<td>On directories, restricted deletion flag.</td>
</tr>
</tbody>
</table>

The user ID of the file is initialised to the effective user ID of the process. The group ID of the file is initialised to either the effective group ID of the process or the group ID of the parent directory.

The owner, group, and other permission bits of `mode` are modified by the file mode creation mask of the process. The `mknod()` function clears each bit whose corresponding bit in the file mode creation mask of the process is set.

Upon successful completion, `mknod()` marks for update the `st_atime`, `st_ctime` and `st_mtime` fields of the file. Also, the `st_ctime` and `st_mtime` fields of the directory that contains the new entry are...
marked for update.

Only a process with appropriate privileges may invoke \texttt{mknod()} for file types other than FIFO-special.

RETURN VALUE

Upon successful completion, \texttt{mknod()} returns 0. Otherwise, it returns \(-1\), the new file is not created, and \texttt{errno} is set to indicate the error.

ERRORS

The \texttt{mknod()} function will fail if:

\begin{itemize}
  \item \textbf{[EPERM]} The invoking process does not have appropriate privileges and the file type is not FIFO-special.
  \item \textbf{[ENOTDIR]} A component of the path prefix is not a directory.
  \item \textbf{[ENOENT]} A component of the path prefix specified by \texttt{path} does not name an existing directory or \texttt{path} is an empty string.
  \item \textbf{[EACCES]} A component of the path prefix denies search permission, or write permission is denied on the parent directory.
  \item \textbf{[EROFS]} The directory in which the file is to be created is located on a read-only file system.
  \item \textbf{[EEXIST]} The named file exists.
  \item \textbf{[EIO]} An I/O error occurred while accessing the file system.
  \item \textbf{[EINVAL]} An invalid argument exists.
  \item \textbf{[ENOSPC]} The directory that would contain the new file cannot be extended or the file system is out of file allocation resources.
  \item \textbf{[ELOOP]} Too many symbolic links were encountered in resolving \texttt{path}.
  \item \textbf{[ENAMETOOLONG]} The length of a pathname exceeds \texttt{PATH_MAX}, or pathname component is longer than \texttt{NAME_MAX}.
\end{itemize}

The \texttt{mknod()} function may fail if:

\begin{itemize}
  \item \textbf{[ENAMETOOLONG]} Pathname resolution of a symbolic link produced an intermediate result whose length exceeds \texttt{PATH_MAX}.
\end{itemize}

APPLICATION USAGE

For portability to implementations conforming to earlier versions of this document, \texttt{mkfifo()} is preferred over this function for making FIFO special files.

SEE ALSO

\texttt{chmod()}, \texttt{creat()}, \texttt{exec}, \texttt{mkdir()}, \texttt{mkfifo()}, \texttt{open()}, \texttt{stat()}, \texttt{umask()}, \texttt{<sys/stat.h>}.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
mkstemp — make a unique file name

SYNOPSIS

```c
#include <stdlib.h>

int mkstemp(char *template);
```

DESCRIPTION
The `mkstemp()` function replaces the contents of the string pointed to by `template` by a unique file name, and returns a file descriptor for the file open for reading and writing. The function thus prevents any possible race condition between testing whether the file exists and opening it for use. The string in `template` should look like a file name with six trailing 'X's; `mkstemp()` replaces each 'X' with a character from the portable file name character set. The characters are chosen such that the resulting name does not duplicate the name of an existing file.

RETURN VALUE
Upon successful completion, `mkstemp()` returns an open file descriptor. Otherwise −1 is returned if no suitable file could be created.

ERRORS
No errors are defined.

APPLICATION USAGE
It is possible to run out of letters.

The `mkstemp()` function does not check to determine whether the file name part of `template` exceeds the maximum allowable file name length.

For portability with previous versions of this document, `tmpfile()` is preferred over this function.

SEE ALSO
`getpid()`, `open()`, `tmpfile()`, `tmpnam()`, `<stdlib.h>`.

CHANGE HISTORY
First released in Issue 4, Version 2.
**NAME**

mktemp — make a unique filename

**SYNOPSIS**

```c
#include <stdlib.h>

char *mktemp(char *template);
```

**DESCRIPTION**

The `mktemp()` function replaces the contents of the string pointed to by `template` by a unique filename and returns `template`. The application must initialise `template` to be a filename with six trailing 'X's; `mktemp( )` replaces each 'X' with a single byte character from the portable filename character set.

**RETURN VALUE**

The `mktemp()` function returns the pointer `template`. If a unique name cannot be created, `template` points to a null string.

**ERRORS**

No errors are defined.

**APPLICATION USAGE**

Between the time a pathname is created and the file opened, it is possible for some other process to create a file with the same name. The `mkstemp()` function avoids this problem.

For portability with previous versions of this document, `tmpnam()` is preferred over this function.

**SEE ALSO**

`mkstemp()`, `tmpfile()`, `tmpnam()`, `<stdlib.h>`.

**CHANGE HISTORY**

First released in Issue 4, Version 2.
NAME
mktime — convert broken-down time into time since the Epoch

SYNOPSIS
#include <time.h>

time_t mktime(struct tm *timeptr);

DESCRIPTION
The mktime() function converts the broken-down time, expressed as local time, in the structure
pointed to by timeptr, into a time since the Epoch value with the same encoding as that of the
values returned by time(). The original values of the tm_wday and tm_yday components of the
structure are ignored, and the original values of the other components are not restricted to the
ranges described in the <time.h> entry.

A positive or 0 value for tm_isdst causes mktime() to presume initially that Daylight Savings
Time, respectively, is or is not in effect for the specified time. A negative value for tm_isdst
causes mktime() to attempt to determine whether Daylight Saving Time is in effect for the
specified time.

Local timezone information is set as though mktime() called tzset().

Upon successful completion, the values of the tm_wday and tm_yday components of the structure
are set appropriately, and the other components are set to represent the specified time since the
Epoch, but with their values forced to the ranges indicated in the <time.h> entry; the final value
of tm_mday is not set until tm_mon and tm_year are determined.

RETURN VALUE
The mktime() function returns the specified time since the Epoch encoded as a value of type
time_t. If the time since the Epoch cannot be represented, the function returns the value
(time_t)−1.

ERRORS
No errors are defined.
EXAMPLES
What day of the week is July 4, 2001?

```c
#include <stdio.h>
#include <time.h>

struct tm time_str;
char daybuf[20];

int main(void)
{
    time_str.tm_year = 2001 - 1900;
    time_str.tm_mon = 7 - 1;
    time_str.tm_mday = 4;
    time_str.tm_hour = 0;
    time_str.tm_min = 0;
    time_str.tm_sec = 1;
    time_str.tm_isdst = -1;
    if (mktime(&time_str) == -1)
        (void)puts("-unknown-");
    else {
        (void)strftime(daybuf, sizeof(daybuf), "%A", &time_str);
        (void)puts(daybuf);
    }
    return 0;
}
```

SEE ALSO
asctime(), clock(), ctime(), difftime(), gmtime(), localtime(), strftime(), strptime(), time(), utime(), <time.h>.

CHANGE HISTORY
First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard and ANSI C standard.

Issue 4
The following changes are incorporated in this issue:

- In the DESCRIPTION section, a paragraph is added indicating the possible settings of tm_isdst, and reference to setting of tm_sec for leap seconds or double leap seconds is removed (although this functionality is still supported).
- In the EXAMPLES section, the sample code is updated to use ISO C syntax.
mmap( )

NAME

mmap — map pages of memory

SYNOPSIS

UX

#include <sys/mman.h>

void *mmap(void *addr, size_t len, int prot, int flags,
int fildes, off_t off);

DESCRIPTION

The mmap() function establishes a mapping between a process' address space and a file. The
format of the call is as follows:

   pa = mmap(addr, len, prot, flags, fildes, off);

The mmap() function establishes a mapping between the process' address space at an address pa
for len bytes and the file associated with the file descriptor fildes at offset off for len bytes. The
value of pa is an unspecified function of the argument addr and values of flags, further described
below. A successful mmap() call returns pa as its result. The address ranges covered by [pa, pa + len) and [off, off + len) must be legitimate for the possible (not necessarily current) address space
of a process and the file, respectively.

If the size of the mapped file changes after the call to mmap(), the effect of references to portions
of the mapped region that correspond to added or removed portions of the file is unspecified.

The mmap() function is supported for regular files. Support for any other type of file is
unspecified.

The prot argument determines whether read, write, execute, or some combination of accesses are
permitted to the pages being mapped. The protection options are defined in <sys/mman.h>:

PROT_READ     Page can be read.
PROT_WRITE    Page can be written.
PROT_EXEC     Page can be executed.
PROT_NONE     Page cannot be accessed.

Implementations need not enforce all combinations of access permissions. However, writes
shall only be permitted when PROT_WRITE has been set.

The flags argument provides other information about the handling of the mapped pages. The
options are defined in <sys/mman.h>:

MAP_SHARED   Share changes.
MAP_PRIVATE  Changes are private.
MAP_FIXED    Interpret addr exactly.

The MAP_PRIVATE and MAP_SHARED flags control the visibility of write references to the
memory region. Exactly one of these flags must be specified. The mapping type is retained
across a fork().

If MAP_SHARED is set in flags, write references to the memory region by the calling process
may change the file and are visible in all MAP_SHARED mappings of the same portion of the
file by any process.

If MAP_PRIVATE is set in flags, write references to the memory region by the calling process do
not change the file and are not visible to any process in other mappings of the same portion of
the file.

It is unspecified whether write references by processes that have mapped the memory region
using MAP_SHARED are visible to processes that have mapped the same portion of the file.
using MAP_PRIVATE.

It is also unspecified whether write references to a memory region mapped with MAP_SHARED are visible to processes reading the file and whether writes to a file are visible to processes that have mapped the modified portion of that file, except for the effect of msync().

When MAP_FIXED is set in the flags argument, the implementation is informed that the value of pa must be addr, exactly. If MAP_FIXED is set, mmap() may return (void *)-1 and set errno to [EINVAL]. If a MAP_FIXED request is successful, the mapping established by mmap() replaces any previous mappings for the process’ pages in the range [pa, pa + len).

When MAP_FIXED is not set, the implementation uses addr in an unspecified manner to arrive at pa. The pa so chosen will be an area of the address space which the implementation deems suitable for a mapping of len bytes to the file. All implementations interpret an addr value of 0 as granting the implementation complete freedom in selecting pa, subject to constraints described below. A non-zero value of addr is taken to be a suggestion of a process address near which the mapping should be placed. When the implementation selects a value for pa, it never places a mapping at address 0, nor does it replace any extant mapping, nor map into dynamic memory allocation areas.

The off argument is constrained to be aligned and sized according to the value returned by sysconf() when passed _SC_PAGESIZE or _SC_PAGE_SIZE. When MAP_FIXED is specified, the argument addr must also meet these constraints. The implementation performs mapping operations over whole pages. Thus, while the argument len need not meet a size or alignment constraint, the implementation will include, in any mapping operation, any partial page specified by the range [pa, pa + len).

The implementation always zero-fills any partial page at the end of a memory region. Further, the implementation never writes out any modified portions of the last page of a file that are beyond the end of the mapped portion of the file. If the mapping established by mmap() extends into pages beyond the page containing the last byte of the file, an application reference to any of the pages in the mapping that are beyond the last page results in the delivery of a SIGBUS or SIGSEGV signal.

The mmap() function adds an extra reference to the file associated with the file descriptor fildes which is not removed by a subsequent close() on that file descriptor. This reference is removed when there are no more mappings to the file.

The st_atime field of the mapped file may be marked for update at any time between the mmap() call and the corresponding munmap() call. The initial read or write reference to a mapped region will cause the file’s st_atime field to be marked for update if it has not already been marked for update.

The st_ctime and st_mtime fields of a file that is mapped with MAP_SHARED and PROT_WRITE, will be marked for update at some point in the interval between a write reference to the mapped region and the next call to msync() with MS_ASYNC or MS_SYNC for that portion of the file by any process. If there is no such call, these fields may be marked for update at any time after a write reference if the underlying file is modified as a result.

There may be implementation-dependent limits on the number of memory regions that can be mapped (per process or per system). If such a limit is imposed, whether the number of memory regions that can be mapped by a process is decreased by the use of shmat() is implementation-dependent.

**RETURN VALUE**

Upon successful completion, mmap() returns the address at which the mapping was placed (pa). Otherwise, it returns a value of −1 and sets errno to indicate the error.
ERRORS

The `mmap()` function will fail if:

- **[EBADF]** The `fildes` argument is not a valid open file descriptor.
- **[EACCES]** The `fildes` argument is not open for read, regardless of the protection specified, or `fildes` is not open for write and PROT_WRITE was specified for a MAP_SHARED type mapping.
- **[ENXIO]** Addresses in the range `[off, off + len)` are invalid for `fildes`.
- **[EINVAL]** The `addr` argument (if MAP_FIXED was specified) or `off` is not a multiple of the page size as returned by `sysconf()`, or are considered invalid by the implementation.
- **[EINVAL]** The value of `flags` is invalid (neither MAP_PRIVATE nor MAP_SHARED is set).
- **[EMFILE]** The number of mapped regions would exceed an implementation-dependent limit (per process or per system).
- **[ENODEV]** The `fildes` argument refers to a file whose type is not supported by `mmap()`.
- **[ENOMEM]** MAP_FIXED was specified, and the range `[addr, addr + len)` exceeds that allowed for the address space of a process; or if MAP_FIXED was not specified and there is insufficient room in the address space to effect the mapping.

APPLICATION USAGE

Use of `mmap()` may reduce the amount of memory available to other memory allocation functions.

Use of MAP_FIXED may result in unspecified behaviour in further use of `brk()`, `sbrk()`, `malloc()` and `shmat()`. The use of MAP_FIXED is discouraged, as it may prevent an implementation from making the most effective use of resources.

The application must ensure correct synchronisation when using `mmap()` in conjunction with any other file access method, such as `read()` and `write()`, standard input/output, and `shmat`.

The `mmap()` function allows access to resources via address space manipulations, instead of `read()/write()`. Once a file is mapped, all a process has to do to access it is use the data at the address to which the file was mapped. So, using pseudo-code to illustrate the way in which an existing program might be changed to use `mmap()`, the following:

```c
fildes = open(...)
lseek(fildes, some_offset)
read(fildes, buf, len)
/* use data in buf */
```

becomes:

```c
fildes = open(...)
address = mmap(0, len, PROT_READ, MAP_PRIVATE, fildes, some_offset)
/* use data at address */
```

SEE ALSO

`exec`, `fcntl()`, `fork()`, `lockf()`, `msync()`, `munmap()`, `mprotect()`, `shmat()`, `sysconf()`, `<sys/mman.h>`.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
modf — decompose floating-point number

SYNOPSIS
#include <math.h>

double modf(double x, double *iptr);

DESCRIPTION
The modf() function breaks the argument x into integral and fractional parts, each of which has
the same sign as the argument. It stores the integral part as a double in the object pointed to by
iptr.

RETURN VALUE
Upon successful completion, modf() returns the signed fractional part of x.

EX
If x is NaN, NaN is returned, errno may be set to [EDOM] and *iptr is set to NaN.
If the correct value would cause underflow, 0 is returned and errno may be set to [ERANGE].

ERRORS
The modf() function may fail if:

EX
[EDOM] The value of x is NaN.
[ERANGE] The result underflows.

EX
No other errors will occur.

APPLICATION USAGE
An application wishing to check for error situations should set errno to 0 before calling modf(). If
errno is non-zero on return, or the return value is NaN, an error has occurred.

SEE ALSO
frexp(), isnan(), ldexp(), <math.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:
• Removed references to matherr().
• The name of the first argument is changed from value to x.
• The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with
the ISO C standard and to rationalise error handling in the mathematics functions.
• The return value specified for [EDOM] is marked as an extension.
mprotect() X/Open UNIX System Interfaces

NAME
mprotect — set protection of memory mapping

SYNOPSIS
UX

#include <sys/mman.h>

int mprotect(void *addr, size_t len, int prot);

DESCRIPTION
The mprotect() function changes the access protections on the mappings specified by the range [addr, addr + len), rounding len up to the next multiple of the page size as returned by sysconf(), to be that specified by prot. Legitimate values for prot are the same as those permitted for mmap() and are defined in <sys/mman.h>:

PROT_READ Page can be read.
PROT_WRITE Page can be written.
PROT_EXEC Page can be executed.
PROT_NONE Page cannot be accessed.

When mprotect() fails for reasons other than [EINVAL], the protections on some of the pages in the range [addr, addr + len) may have been changed.

RETURN VALUE
Upon successful completion, mprotect() returns 0. Otherwise, it returns −1 and sets errno to indicate the error.

ERRORS
The mprotect() function will fail if:

[EACCES] The prot argument specifies a protection that violates the access permission the process has to the underlying memory object.

[EINVAL] The addr argument is not a multiple of the page size as returned by sysconf().

ENOMEM Addresses in the range [addr, addr + len) are invalid for the address space of a process, or specify one or more pages which are not mapped.

The mprotect() function may fail if:

[EAGAIN] The prot argument specifies PROT_WRITE over a MAP_PRIVATE mapping and there are insufficient memory resources to reserve for locking the private page.

SEE ALSO
mmap(), sysconf(), <sys/mman.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
mrand48 — generate uniformly distributed pseudo-random signed long integers

SYNOPSIS
EX
#include <stdlib.h>
long int mrand48(void);

DESCRIPTION
Refer to drand48().

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:
• The <stdlib.h> header is now required.
• The mrand48() function is now defined to return long int.
• The argument list now includes void.
NAME

msgctl — message control operations

SYNOPSIS

```c
#include <sys/msg.h>

int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

DESCRIPTION

The `msgctl()` function provides message control operations as specified by `cmd`. The following values for `cmd`, and the message control operations they specify, are:

- **IPC_STAT**
  Place the current value of each member of the `msqid_ds` data structure associated with `msqid` into the structure pointed to by `buf`. The contents of this structure are defined in `<sys/msg.h>`.

- **IPC_SET**
  Set the value of the following members of the `msqid_ds` data structure associated with `msqid` to the corresponding value found in the structure pointed to by `buf`:
    ```
    msg_perm.uid
    msg_perm.gid
    msg_perm.mode
    msg_qbytes
    ```

  IPC_SET can only be executed by a process with appropriate privileges or that has an effective user ID equal to the value of `msg_perm.cuid` or `msg_perm.uid` in the `msqid_ds` data structure associated with `msqid`. Only a process with appropriate privileges can raise the value of `msg_qbytes`.

- **IPC_RMID**
  Remove the message queue identifier specified by `msqid` from the system and destroy the message queue and `msqid_ds` data structure associated with it. IPC_RMD can only be executed by a process with appropriate privileges or one that has an effective user ID equal to the value of `msg_perm.cuid` or `msg_perm.uid` in the `msqid_ds` data structure associated with `msqid`.

RETURN VALUE

Upon successful completion, `msgctl()` returns 0. Otherwise, it returns −1 and `errno` will be set to indicate the error.

ERRORS

The `msgctl()` function will fail if:

- **[EACCES]** The argument `cmd` is IPC_STAT and the calling process does not have read permission, see Section 2.6 on page 37.

- **[EINVAL]** The value of `msqid` is not a valid message queue identifier; or the value of `cmd` is not a valid command.

- **[EPERM]** The argument `cmd` is IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of `msg_perm.cuid` or `msg_perm.uid` in the data structure associated with `msqid`.

- **[EPERM]** The argument `cmd` is IPC_SET, an attempt is being made to increase to the value of `msg_qbytes`, and the effective user ID of the calling process does not have appropriate privileges.
FUTURE DIRECTIONS
The IEEE 1003.4 Standards Committee is developing alternative interfaces for interprocess Communication. Application developers who need to use IPC should design their applications so that modules using the routines described in this document can be easily modified to use alternative methods at a later date.

SEE ALSO
msgget(), msgrcv(), msgsnd(), <sys/msg.h>, Section 2.6 on page 37.

CHANGE HISTORY
First released in Issue 2.
Derived from Issue 2 of the SVID.

Issue 4
The following changes are incorporated in this issue:

- The interface is no longer marked as OPTIONAL FUNCTIONALITY.
- Inclusion of the <sys/types.h> and <sys/ipc.h> headers is removed from the SYNOPSIS section.
- A FUTURE DIRECTIONS section is added warning application developers about migration to IEEE 1003.4 interfaces for interprocess communication.
- The [ENOSYS] error is removed from the ERRORS section.
msgget() — get message queue

SYNOPSIS

```c
#include <sys/msg.h>

int msgget(key_t key, int msgflg);
```

DESCRIPTION

The `msgget()` function returns the message queue identifier associated with the argument `key`.

A message queue identifier, associated message queue and data structure, see `<sys/msg.h>`, are created for the argument `key` if one of the following is true:

- The argument `key` is equal to IPC_PRIVATE.
- The argument `key` does not already have a message queue identifier associated with it, and `(msgflg & IPC_CREAT)` is non-zero.

Upon creation, the data structure associated with the new message queue identifier is initialised as follows:

- `msg_perm.cuid`, `msg_perm.uid`, `msg_perm.cgid` and `msg_perm.gid` are set equal to the effective user ID and effective group ID, respectively, of the calling process.
- The low-order 9 bits of `msg_perm.mode` are set equal to the low-order 9 bits of `msgflg`.
- `msg_qnum`, `msg_lspid`, `msg_lrpid`, `msg_stime` and `msg_rtime` are set equal to 0.
- `msg_ctime` is set equal to the current time.
- `msg_qbytes` is set equal to the system limit.

RETURN VALUE

Upon successful completion, `msgget()` returns a non-negative integer, namely a message queue identifier. Otherwise, it returns −1 and `errno` is set to indicate the error.

ERRORS

The `msgget()` function will fail if:

- **[EACCES]** A message queue identifier exists for the argument `key`, but operation permission as specified by the low-order 9 bits of `msgflg` would not be granted, see Section 2.6 on page 37.
- **[EEXIST]** A message queue identifier exists for the argument `key` but `((msgflg & IPC_CREAT) && (msgflg & IPC_EXCL))` is non-zero.
- **[ENOENT]** A message queue identifier does not exist for the argument `key` and `((msgflg & IPC_CREAT) == 0)`.
- **[ENOSPC]** A message queue identifier is to be created but the system-imposed limit on the maximum number of allowed message queue identifiers system-wide would be exceeded.

FUTURE DIRECTIONS

The IEEE 1003.4 Standards Committee is developing alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the routines described in this document can be easily modified to use alternative methods at a later date.

SEE ALSO

`msgctl()`, `msgcrm()`, `msgsnd()`, `<sys/msg.h>`, Section 2.6 on page 37.
CHANGE HISTORY
First released in Issue 2.
Derived from Issue 2 of the SVID.

Issue 4
The following changes are incorporated in this issue:

- The interface is no longer marked as OPTIONAL FUNCTIONALITY.
- Inclusion of the <sys/types.h> and <sys/ipc.h> headers is removed from the SYNOPSIS section.
- The [ENOSYS] error is removed from the ERRORS section.
NAME

msgrcv — message receive operation

SYNOPSIS

```c
#include <sys/msg.h>

int msgrcv(int msqid, void *msgp, size_t msgsz, long int msgtyp,
            int msgflg);
```

DESCRIPTION

The `msgrcv()` function reads a message from the queue associated with the message queue identifier specified by `msqid` and places it in the user-defined buffer pointed to by `msgp`.

The argument `msgp` points to a user-defined buffer that must contain first a field of type `long int` that will specify the type of the message, and then a data portion that will hold the data bytes of the message. The structure below is an example of what this user-defined buffer might look like:

```c
struct mymsg {
    long int mtype; /* message type */
    char mtext[1]; /* message text */
};
```

The structure member `mtype` is the received message’s type as specified by the sending process. The structure member `mtext` is the text of the message.

The argument `msgsz` specifies the size in bytes of `mtext`. The received message is truncated to `msgsz` bytes if it is larger than `msgsz` and `(msgflg & MSG_NOERROR)` is non-zero. The truncated part of the message is lost and no indication of the truncation is given to the calling process.

The argument `msgtyp` specifies the type of message requested as follows:

- If `msgtyp` is 0, the first message on the queue is received.
- If `msgtyp` is greater than 0, the first message of type `msgtyp` is received.
- If `msgtyp` is less than 0, the first message of the lowest type that is less than or equal to the absolute value of `msgtyp` is received.

The argument `msgflg` specifies the action to be taken if a message of the desired type is not on the queue. These are as follows:

- If `(msgflg & IPC_NOWAIT)` is non-zero, the calling process will return immediately with a return value of −1 and `errno` set to [ENOMSG].
- If `(msgflg & IPC_NOWAIT)` is 0, the calling process will suspend execution until one of the following occurs:
  - A message of the desired type is placed on the queue.
  - The message queue identifier `msqid` is removed from the system; when this occurs, `errno` is set equal to [EIDRM] and −1 is returned.
  - The calling process receives a signal that is to be caught; in this case a message is not received and the calling process resumes execution in the manner prescribed in `sigaction()`.

X/Open CAE Specification (1994)
Upon successful completion, the following actions are taken with respect to the data structure associated with \textit{msqid}:

\begin{itemize}
  \item \textit{msg_qnum} is decremented by 1.
  \item \textit{msg_lrpid} is set equal to the process ID of the calling process.
  \item \textit{msg_rtime} is set equal to the current time.
\end{itemize}

\textbf{RETURN VALUE}

Upon successful completion, \texttt{msgrcv()} returns a value equal to the number of bytes actually placed into the buffer \textit{mtext}. Otherwise, no message will be received, \texttt{msgrcv()} will return \texttt{-1} and \texttt{errno} will be set to indicate the error.

\textbf{ERRORS}

The \texttt{msgrcv()} function will fail if:

\begin{itemize}
  \item \texttt{[E2BIG]} The value of \textit{mtext} is greater than \textit{msgsz} and \((\textit{msgflg} \& \text{MSG\_NOERROR})\) is 0.
  \item \texttt{[EACCES]} Operation permission is denied to the calling process. See Section 2.6 on page 37.
  \item \texttt{[EIDRM]} The message queue identifier \textit{msqid} is removed from the system.
  \item \texttt{[EINTR]} The \texttt{msgrcv()} function was interrupted by a signal.
  \item \texttt{[EINVAL]} \textit{msqid} is not a valid message queue identifier; or the value of \textit{msgsz} is less than 0.
  \item \texttt{[ENOMSG]} The queue does not contain a message of the desired type and \((\textit{msgflg} \& \text{IPC\_NOWAIT})\) is non-zero.
\end{itemize}

\textbf{APPLICATION USAGE}

The value passed as the \textit{msgp} argument should be converted to type \texttt{void \*}.

\textbf{FUTURE DIRECTIONS}

The IEEE 1003.4 Standards Committee is developing alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the routines described in this document can be easily modified to use alternative methods at a later date.

\textbf{SEE ALSO}

\texttt{msgctl()}, \texttt{msgget()}, \texttt{msgsnd()}, \texttt{sigaction()}, \texttt{<sys/msg.h>}, Section 2.6 on page 37.

\textbf{CHANGE HISTORY}

First released in Issue 2.

Derived from Issue 2 of the SVID.

\textbf{Issue 4}

The following changes are incorporated in this issue:

\begin{itemize}
  \item The interface is no longer marked as OPTIONAL FUNCTIONALITY.
  \item Inclusion of the \texttt{<sys/types.h>} and \texttt{<sys/ipc.h>} headers is removed from the SYNOPSIS section.
  \item The \texttt{[ENOSYS]} error is removed from the ERRORS section.
  \item A FUTURE DIRECTIONS section is added warning application developers about migration to IEEE 1003.4 interfaces for interprocess communication.
\end{itemize}
**NAME**

msgsnd — message send operation

**SYNOPSIS**

```c
#include <sys/msg.h>

int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

**DESCRIPTION**

The `msgsnd()` function is used to send a message to the queue associated with the message queue identifier specified by `msqid`.

The argument `msgp` points to a user-defined buffer that must contain first a field of type `long int` that will specify the type of the message, and then a data portion that will hold the data bytes of the message. The structure below is an example of what this user-defined buffer might look like:

```c
struct mymsg {
  long int mtype; /* message type */
  char mtext[1]; /* message text */
}
```

The structure member `mtype` is a non-zero positive type `long int` that can be used by the receiving process for message selection.

The structure member `mtext` is any text of length `msgsz` bytes. The argument `msgsz` can range from 0 to a system-imposed maximum.

The argument `msgflg` specifies the action to be taken if one or more of the following are true:

- The number of bytes already on the queue is equal to `msg_qbytes`, see `<sys/msg.h>`.
- The total number of messages on all queues system-wide is equal to the system-imposed limit.

These actions are as follows:

- If `(msgflg & IPC_NOWAIT)` is non-zero, the message will not be sent and the calling process will return immediately.
- If `(msgflg & IPC_NOWAIT)` is 0, the calling process will suspend execution until one of the following occurs:
  - The condition responsible for the suspension no longer exists, in which case the message is sent.
  - The message queue identifier `msqid` is removed from the system; when this occurs, `errno` is set equal to `[EIDRM]` and −1 is returned.
  - The calling process receives a signal that is to be caught; in this case the message is not sent and the calling process resumes execution in the manner prescribed in `sigaction()`.

Upon successful completion, the following actions are taken with respect to the data structure associated with `msqid`, see `<sys/msg.h>`:

- `msg_qnum` is incremented by 1.
- `msg_lspid` is set equal to the process ID of the calling process.
- `msg_stime` is set equal to the current time.
RETURN VALUE
Upon successful completion, \texttt{msgsnd()} returns 0. Otherwise, no message will be sent, \texttt{msgsnd()} will return \(-1\) and \texttt{errno} will be set to indicate the error.

ERRORS
The \texttt{msgsnd()} function will fail if:

\begin{itemize}
\item \textbf{[EACCES]} Operation permission is denied to the calling process. See Section 2.6 on page 37.
\item \textbf{[EAGAIN]} The message cannot be sent for one of the reasons cited above and \((\texttt{msgflg} \& \texttt{IPC\_NOWAIT})\) is non-zero.
\item \textbf{[EIDRM]} The message queue identifier \texttt{msgid} is removed from the system.
\item \textbf{[EINTR]} The \texttt{msgsnd()} function was interrupted by a signal.
\item \textbf{[EINVAL]} The value of \texttt{msqid} is not a valid message queue identifier, or the value of \texttt{mtype} is less than \(1\); or the value of \texttt{msgsz} is less than \(0\) or greater than the system-imposed limit.
\end{itemize}

APPLICATION USAGE
The value passed as the \texttt{msgp} argument should be converted to type \texttt{void *}.

FUTURE DIRECTIONS
The IEEE 1003.4 Standards Committee is developing alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the routines described in this document can be easily modified to use alternative methods at a later date.

SEE ALSO
\texttt{msgctl()}, \texttt{msgget()}, \texttt{msgrecv()}, \texttt{sigaction()}, \texttt{<sys/msg.h>}, Section 2.6 on page 37.

CHANGE HISTORY
First released in Issue 2.
Derived from Issue 2 of the SVID.

Issue 4
The following changes are incorporated in this issue:

\begin{itemize}
\item The interface is no longer marked as OPTIONAL FUNCTIONALITY.
\item Inclusion of the \texttt{<sys/types.h>} and \texttt{<sys/ipc.h>} headers is removed from the SYNOPSIS section. Also the type of argument \texttt{msgp} is changed from \texttt{void*} to \texttt{const void*}.
\item In the DESCRIPTION section, the example of a message buffer is changed:
\begin{itemize}
\item explicitly to define the first member as being of type \texttt{long int}
\item to define the size of the message array \texttt{mtext}.
\end{itemize}
\item The [ENOSYS] error is removed from the ERRORS section.
\item A FUTURE DIRECTIONS section is added warning application developers about migration to IEEE 1003.4 interfaces for interprocess communication.
\end{itemize}
msync()  X/OPEN UNIX  System Interfaces

NAME
msync — synchronise memory with physical storage

SYNOPSIS

UX
#include <sys/mman.h>

int msync(void *addr, size_t len, int flags);

DESCRIPTION
The msync() function writes all modified copies of pages over the range [addr, addr + len) to the underlying hardware, or invalidates any copies so that further references to the pages will be obtained by the system from their permanent storage locations.

The flags argument is one of the following:

MS_ASYNC perform asynchronous writes
MS_SYNC perform synchronous writes
MS_INVALIDATE invalidate mappings

If flags is MS_ASYNC or MS_SYNC, the function synchronises the file contents to match the current contents of the memory region.

• All write references to the memory region made prior to the call are visible by subsequent read operations on the file.
• It is unspecified whether writes to the same portion of the file prior to the call are visible by read references to the memory region.
• It is unspecified whether unmodified pages in the specified range are also written to the underlying hardware.

If flags is MS_ASYNC, the function may return immediately once all write operations are scheduled; if flags is MS_SYNC, the function does not return until all write operations are completed.

If flags is MS_INVALIDATE, the function synchronises the contents of the memory region to match the current file contents.

• All writes to the mapped portion of the file made prior to the call are visible by subsequent read references to the mapped memory region.
• It is unspecified whether write references prior to the call, by any process, to memory regions mapped to the same portion of the file using MAP_SHARED, are visible by read references to the region.

If msync() causes any write to the file, then the file’s st_ctime and st_mtime fields are marked for update.

RETURN VALUE
Upon successful completion, msync() returns 0. Otherwise, it returns −1 and sets errno to indicate the error.

ERRORS
The msync() function will fail if:

[EINVAL] The addr argument is not a multiple of the page size as returned by sysconf().
[EIO] An I/O error occurred while reading from or writing to the file system.
[ENOMEM] Some or all the addresses in the range [addr, addr + len) are invalid for the address space of the process or pages not mapped are specified.
APPLICATION USAGE

The `msync()` function should be used by programs that require a memory object to be in a known state, for example in building transaction facilities.

Normal system activity can cause pages to be written to disk. Therefore, there are no guarantees that `msync()` is the only control over when pages are or are not written to disk.

SEE ALSO

`mmap()`, `sysconf()`, `<sys/mman.h>`.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
munmap — unmap pages of memory

SYNOPSIS
#include <sys/mman.h>

int munmap(void *addr, size_t len);

DESCRIPTION
The munmap() function removes the mappings for pages in the range [addr, addr + len), rounding the len argument up to the next multiple of the page size as returned by sysconf(). If addr is not the address of a mapping established by a prior call to mmap(), the behaviour is undefined. After a successful call to munmap() and before any subsequent mapping of the unmapped pages, further references to these pages will result in the delivery of a SIGBUS or SIGSEGV signal to the process.

RETURN VALUE
Upon successful completion, munmap() returns 0. Otherwise, it returns −1 and sets errno to indicate the error.

ERRORS
The munmap() function will fail if:

- [EINVAL] The addr argument is not a multiple of the page size as returned by sysconf().
- [EINVAL] Addresses in the range [addr, addr + len) are outside the valid range for the address space of a process.
- [EINVAL] The len argument is 0.

SEE ALSO
mmap(), sysconf(), <signal.h>, <sys/mman.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
nextafter — next representable double-precision floating-point number

SYNOPSIS
UX
#include <math.h>

double nextafter(double x, double y);

DESCRIPTION
The nextafter() function computes the next representable double-precision floating-point value following x in the direction of y. Thus, if y is less than x, nextafter() returns the largest representable floating-point number less than x.

RETURN VALUE
The nextafter() function returns the next representable double-precision floating-point value following x in the direction of y.

If x or y is NaN, then nextafter() returns NaN and may set errno to [EDOM].

If x is finite and the correct function value would overflow, HUGE_VAL is returned and errno is set to [ERANGE].

ERRORS
The nextafter() function will fail if:
[ERANGE] The correct value would overflow.

The nextafter() function may fail if:
[EDOM] The x or y argument is NaN.

SEE ALSO
<math.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME

nftw — walk a file tree

SYNOPSIS

```c
#include <ftw.h>

int nftw(const char *path,
          int (*fn)(const char *, const struct stat *, int, struct FTW *),
          int depth, int flags);
```

DESCRIPTION

The \texttt{nftw}() function recursively descends the directory hierarchy rooted in \texttt{path}. The \texttt{nftw}() function has a similar effect to \texttt{ftw}() except that it takes an additional argument \texttt{flags}, which is a bitwise inclusive-OR of zero or more of the following flags:

- **FTW\_CHDIR** If set, \texttt{nftw}() will change the current working directory to each directory as it reports files in that directory. If clear, \texttt{nftw}() will not change the current working directory.

- **FTW\_DEPTH** If set, \texttt{nftw}() will report all files in a directory before reporting the directory itself. If clear, \texttt{nftw}() will report any directory before reporting the files in that directory.

- **FTW\_MOUNT** If set, \texttt{nftw}() will only report files in the same file system as \texttt{path}. If clear, \texttt{nftw}() will report all files encountered during the walk.

- **FTW\_PHYS** If set, \texttt{nftw}() performs a physical walk and does not follow symbolic links. If clear, \texttt{nftw}() will follow links instead of reporting them, and will not report the same file twice.

At each file it encounters, \texttt{nftw}() calls the user-supplied function \texttt{fn}() with four arguments:

- The first argument is the pathname of the object.

- The second argument is a pointer to the \texttt{stat} buffer containing information on the object.

- The third argument is an integer giving additional information. Its value is one of the following:
  - **FTW\_F** The object is a file.
  - **FTW\_D** The object is a directory.
  - **FTW\_DP** The object is a directory and subdirectories have been visited. (This condition will only occur if the FTW\_DEPTH flag is included in \texttt{flags}.)
  - **FTW\_SL** The object is a symbolic link. (This condition will only occur if the FTW\_PHYS flag is included in \texttt{flags}.)
  - **FTW\_SLN** The object is a symbolic link that does not name an existing file. (This condition will only occur if the FTW\_PHYS flag is not included in \texttt{flags}.)
  - **FTW\_DNR** The object is a directory that cannot be read. The \texttt{fn}() function will not be called for any of its descendants.
  - **FTW\_NS** The \texttt{stat}() function failed on the object because of lack of appropriate permission. The \texttt{stat} buffer passed to \texttt{fn}() is undefined. Failure of \texttt{stat}() for any other reason is considered an error and \texttt{nftw}() returns \texttt{-1}.

- The fourth argument is a pointer to an \texttt{FTW} structure. The value of \texttt{base} is the offset of the object's filename in the pathname passed as the first argument to \texttt{fn}(). The value of \texttt{level} indicates depth relative to the root of the walk, where the root level is 0.
The argument *depth* limits the directory depth for the search. At most one file descriptor will be used for each directory level.

**RETURN VALUE**

The *nftw()* function continues until the first of the following conditions occurs:

- An invocation of *fn()* returns a non-zero value, in which case *nftw()* returns that value.
- The *nftw()* function detects an error other than [EACCES] (see FTW_DNR and FTW_NS above), in which case *nftw()* returns −1 and sets *errno* to indicate the error.
- The tree is exhausted, in which case *nftw()* returns 0.

**ERRORS**

The *nftw()* function will fail if:

- [EACCES] Search permission is denied for any component of *path* or read permission is denied for *path*, or *fn()* returns −1 and does not reset *errno*.
- [ENAMETOOLONG] The length of the *path* string exceeds {PATH_MAX}, or a pathname component is longer than {NAME_MAX}.
- [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- [ENOTDIR] A component of *path* is not a directory.

The *nftw()* function may fail if:

- [ELOOP] Too many symbolic links were encountered in resolving *path*.
- [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.
- [ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.
- [ENFILE] Too many files are currently open in the system.

In addition, *errno* may be set if the function pointed by *fn()* causes *errno* to be set.

**SEE ALSO**

*lstat*(), *opendir*(), *readdir*(), *stat*(), <*ftw.h*>.

**CHANGE HISTORY**

First released in Issue 4, Version 2.
NAME
nice — change priority of a process

SYNOPSIS
EX
#include <unistd.h>

int nice(int incr);

DESCRIPTION
The nice() function adds the value of incr to the nice value of the calling process. A process' nice value is a non-negative number for which a more positive value results in lower CPU priority.

A maximum nice value of 2 * {NZERO} -1 and a minimum nice value of 0 are imposed by the system. Requests for values above or below these limits result in the nice value being set to the corresponding limit. Only a process with appropriate privileges can lower the nice value.

RETURN VALUE
Upon successful completion, nice() returns the new nice value minus {NZERO}. Otherwise, -1 is returned, the process' nice value is not changed, and errno is set to indicate the error.

ERRORS
The nice() function will fail if:

[EPERM] The incr argument is negative and the calling process does not have appropriate privileges.

APPLICATION USAGE
As -1 is a permissible return value in a successful situation, an application wishing to check for error situations should set errno to 0, then call nice(), and if it returns -1, check to see if errno is non-zero.

SEE ALSO
<limits.h>, <unistd.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

• The header <unistd.h> is added to the SYNOPSIS section.

• A statement is added to the DESCRIPTION section indicating that the nice value can only be lowered by a process with appropriate privileges.

Issue 4, Version 2
The RETURN VALUE section is updated for X/OPEN UNIX conformance to define that the process' nice value is not changed if an error is detected.
NAME

nl_langinfo — language information

SYNOPSIS

EX

```c
#include <langinfo.h>

char *nl_langinfo(nl_item item);
```

DESCRIPTION

The `nl_langinfo()` function returns a pointer to a string containing information relevant to the particular language or cultural area defined in the program's locale (see `<langinfo.h>`). The manifest constant names and values of `item` are defined in `<langinfo.h>`. For example:

```c
nl_langinfo(ABDAY_1)
```

would return a pointer to the string “Dom” if the identified language was Portuguese, and “Sun” if the identified language was English.

RETURN VALUE

In a locale where `langinfo` data is not defined, `nl_langinfo()` returns a pointer to the corresponding string in the POSIX locale. In all locales, `nl_langinfo()` returns a pointer to an empty string if `item` contains an invalid setting.

ERRORS

No errors are defined.

APPLICATION USAGE

The array pointed to by the return value should not be modified by the program, but may be modified by further calls to `nl_langinfo()`. In addition, calls to `setlocale()` with a category corresponding to the category of `item` (see `<langinfo.h>`), or to the category LC_ALL, may overwrite the array.

SEE ALSO

`setlocale()`, `<langinfo.h>`, `<nl_types.h>`, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY

First released in Issue 2.

Issue 4

The header `<nl_types.h>` is removed from the SYNOPSIS section.
NAME
   nrand48 — generate uniformly distributed pseudo-random non-negative long integers

SYNOPSIS
   EX
   #include <stdlib.h>
   long int nrand48(unsigned short int xsubi[3]);

DESCRIPTION
   Refer to drand48().

CHANGE HISTORY
   First released in Issue 1.
   Derived from Issue 1 of the SVID.

Issue 4
   The declaration of xsubi is expanded to unsigned short int.
NAME

open — open a file

SYNOPSIS

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *path, int oflag, ...);
```

DESCRIPTION

The `open()` function establishes the connection between a file and a file descriptor. It creates an open file description that refers to a file and a file descriptor that refers to that open file description. The file descriptor is used by other I/O functions to refer to that file. The `path` argument points to a pathname naming the file.

The `open()` function will return a file descriptor for the named file that is the lowest file descriptor not currently open for that process. The open file description is new, and therefore the file descriptor does not share it with any other process in the system. The FD_CLOEXEC file descriptor flag associated with the new file descriptor will be cleared.

The file offset used to mark the current position within the file is set to the beginning of the file.

The file status flags and file access modes of the open file description will be set according to the value of `oflag`.

Values for `oflag` are constructed by a bitwise-inclusive-OR of flags from the following list, defined in `<fcntl.h>`. Applications must specify exactly one of the first three values (file access modes) below in the value of `oflag`:

- `O_RDONLY` Open for reading only.
- `O_WRONLY` Open for writing only.
- `O_RDWR` Open for reading and writing. The result is undefined if this flag is applied to a FIFO.

Any combination of the following may be used:

- `O_APPEND` If set, the file offset will be set to the end of the file prior to each write.
- `O_CREAT` If the file exists, this flag has no effect except as noted under `O_EXCL` below. Otherwise, the file is created; the user ID of the file is set to the effective user ID of the process; the group ID of the file is set to the group ID of the file's parent directory or to the effective group ID of the process; and the access permission bits (see `<sys/stat.h>`) of the file mode are set to the value of the third argument taken as type `mode_t` modified as follows: a bitwise-AND is performed on the file-mode bits and the corresponding bits in the complement of the process' file mode creation mask. Thus, all bits in the file mode whose corresponding bit in the file mode creation mask is set are cleared. When bits other than the file permission bits are set, the effect is unspecified. The third argument does not affect whether the file is open for reading, writing or for both.
- `O_EXCL` If `O_CREAT` and `O_EXCL` are set, `open()` will fail if the file exists. The check for the existence of the file and the creation of the file if it does not exist will be atomic with respect to other processes executing `open()` naming the same filename in the same directory with `O_EXCL` and `O_CREAT` set. If `O_CREAT` is not set, the effect is undefined.
**open()**

If set and *path* identifies a terminal device, *open()* will not cause the terminal device to become the controlling terminal for the process.

**O_NONBLOCK** When opening a FIFO with O_RDONLY or O_WRONLY set:

If O_NONBLOCK is set:

An *open()* for reading only will return without delay. An *open()* for writing only will return an error if no process currently has the file open for reading.

If O_NONBLOCK is clear:

An *open()* for reading only will block until a process opens the file for writing. An *open()* for writing only will block until a process opens the file for reading.

When opening a block special or character special file that supports non-blocking opens:

If O_NONBLOCK is set:

The *open()* function will return without blocking for the device to be ready or available. Subsequent behaviour of the device is device-specific.

If O_NONBLOCK is clear:

The *open()* function will block until the device is ready or available before returning.

Otherwise, the behaviour of O_NONBLOCK is unspecified.

**EX**

**O_SYNC** If O_SYNC is set on a regular file, writes to that file will cause the process to block until the data is delivered to the underlying hardware.

**O_TRUNC** If the file exists and is a regular file, and the file is successfully opened O_RDWR or O_WRONLY, its length is truncated to 0 and the mode and owner are unchanged. It will have no effect on FIFO special files or terminal device files. Its effect on other file types is implementation-dependent. The result of using O_TRUNC with O_RDONLY is undefined.

If O_CREAT is set and the file did not previously exist, upon successful completion, *open()* will mark for update the st_atime, st_ctime and st_mtime fields of the file and the st_ctime and st_mtime fields of the parent directory.

If O_TRUNC is set and the file did previously exist, upon successful completion, *open()* will mark for update the st_ctime and st_mtime fields of the file.

**UX**

If *path* refers to a STREAMS file, *oflag* may be constructed from O_NONBLOCK OR-ed with either O_RDONLY, O_WRONLY, or O_RDWR. Other flag values are not applicable to STREAMS devices and have no effect on them. The value O_NONBLOCK affects the operation of STREAMS drivers and certain functions applied to file descriptors associated with STREAMS files. For STREAMS drivers, the implementation of O_NONBLOCK is device-specific.

If *path* names the master side of a pseudo-terminal device, then it is unspecified whether *open()* locks the slave side so that it cannot be opened. Portable applications must call *unlockpt()* before opening the slave side.

**RETURN VALUE**

Upon successful completion, the function will open the file and return a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, −1 is returned and *errno* is set to indicate the error. No files will be created or modified if the function returns −1.

422 X/Open CAE Specification (1994)
**ERRORS**

The `open()` function will fail if:

- **[EACCES]** Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by `oflag` are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created, or `O_TRUNC` is specified and write permission is denied.
- **[EEXIST]** `O_CREAT` and `O_EXCL` are set, and the named file exists.
- **[EINTR]** A signal was caught during `open()`.
- **[EIO]** The `path` argument names a STREAMS file and a hangup or error occurred during the `open()`.
- **[EISDIR]** The named file is a directory and `oflag` includes `O_WRONLY` or `O_RDWR`.
- **[ELOOP]** Too many symbolic links were encountered in resolving `path`.
- **[EMFILE]** `OPEN_MAX` file descriptors are currently open in the calling process.
- **[ENAMETOOLONG]** The length of the `path` argument exceeds `{PATH_MAX}` or a pathname component is longer than `{NAME_MAX}`.
- **[ENFILE]** The maximum allowable number of files is currently open in the system.
- **[ENOENT]** `O_CREAT` is not set and the named file does not exist; or `O_CREAT` is set and either the path prefix does not exist or the `path` argument points to an empty string.
- **[ENOSR]** The `path` argument names a STREAMS-based file and the system is unable to allocate a STREAM.
- **[ENOSPC]** The directory or file system that would contain the new file cannot be expanded, the file does not exist, and `O_CREAT` is specified.
- **[ENOTDIR]** A component of the path prefix is not a directory.
- **[ENXIO]** `O_NONBLOCK` is set, the named file is a FIFO, `O_WRONLY` is set and no process has the file open for reading.
- **[ERofs]** The named file resides on a read-only file system and either `O_WRONLY`, `O_RDWR`, `O_CREAT` (if file does not exist) or `O_TRUNC` is set in the `oflag` argument.

The `open()` function may fail if:

- **[EAGAIN]** The `path` argument names the slave side of a pseudo-terminal device that is locked.
- **[EINVAL]** The value of the `oflag` argument is not valid.
- **[ENAMETOOLONG]** Pathname resolution of a symbolic link produced an intermediate result whose length exceeds `{PATH_MAX}`.
- **[ENOMEM]** The `path` argument names a STREAMS file and the system is unable to allocate resources.
EX

The file is a pure procedure (shared text) file that is being executed and oflag is O_WRONLY or O_RDWR.

SEE ALSO

chmod(), close(), creat(), dup(), fcntl(), lseek(), read(), umask(), unlockpt(), write(), <fcntl.h>, <sys/stat.h>, <sys/types.h>.

CHANGE HISTORY

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

Issue 4

The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- The type of argument path is changed from char * to const char *.
- Various wording changes are made to the DESCRIPTION section to improve clarity and to align the text with the ISO POSIX-1 standard.

The following changes are incorporated for alignment with the FIPS requirements:

- In the DESCRIPTION section, the description of O_CREAT is amended and the relevant part marked as an extension.
- In the ERRORS section, the condition whereby [ENAMETOOLONG] will be returned if a pathname component is larger that {NAME_MAX} is now defined as mandatory and marked as an extension.

Other changes are incorporated as follows:

- The headers <sys/types.h> and <sys/stat.h> are now marked as optional (OH); these headers do not need to be included on XSI-conformant systems.
- O_NDELAY is removed from the list of oflag values (this flag was marked WITHDRAWN in Issue 3).
- The [ENXIO] error (for the condition where the file is a character or block special file and the associated device does not exist) and the [EINVAL] error are marked as extensions.

Issue 4, Version 2

The following changes are incorporated for X/OPEN UNIX conformance:

- The DESCRIPTION is updated to define the use of open flags with STREAMS files, and to identify special considerations when opening the master side of a pseudo-terminal.
- The [EIO], [ELOOP] and [ENOSR] errors are added to the ERRORS section as mandatory errors; [EAGAIN], [ENAMETOOLONG] and [ENOMEM] are added as optional errors.
NAME
opendir — open directory

SYNOPSIS
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char *dirname);

DESCRIPTION
The opendir() function opens a directory stream corresponding to the directory named by the
dirname argument. The directory stream is positioned at the first entry. If the type DIR, is
implemented using a file descriptor, applications will only be able to open up to a total of
{OPEN_MAX} files and directories. A successful call to any of the exec functions will close any
directory streams that are open in the calling process.

RETURN VALUE
Upon successful completion, opendir() returns a pointer to an object of type DIR. Otherwise, a
null pointer is returned and errno is set to indicate the error.

ERRORS
The opendir() function will fail if:

[EACCES] Search permission is denied for the component of the path prefix of dirname or
read permission is denied for dirname.

[ELOOP] Too many symbolic links were encountered in resolving path.

[ENAMETOOLONG] The length of the dirname argument exceeds {PATH_MAX}, or a pathname
component is longer than {NAME_MAX}.

[ENOENT] A component of dirname does not name an existing directory or dirname is an
empty string.

[ENOTDIR] A component of dirname is not a directory.

The opendir() function may fail if:

[EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result
whose length exceeds {PATH_MAX}.

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

APPLICATION USAGE
The opendir() function should be used in conjunction with readdir(), closedir() and
rewinddir() to examine the contents of the directory (see the EXAMPLES section in readdir()). This method is
recommended for portability.

SEE ALSO
closedir(), lstat(), readdir(), rewinddir(), symlink(), dirent.h>, limits.h>, <sys/types.h>.
CHANGE HISTORY

First released in Issue 2.

Issue 4

The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- The type of argument *dirname* is changed from *char* to *const char*.
- The generation of an [ENOENT] error when *dirname* points to an empty string is made mandatory.

The following change is incorporated for alignment with the FIPS requirements:

- In the *ERRORS* section, the condition whereby [ENAMETOOLONG] will be returned if a pathname component is larger than [NAME_MAX] is now defined as mandatory and marked as an extension.

Other changes are incorporated as follows:

- The header <sys/types.h> is now marked as optional (OH); this header need not be included on XSI-conformant systems.
- In the DESCRIPTION section, the following sentence is moved to the XBD specification:

  The type *DIR*, which is defined in <dirent.h>, represents a *directory stream*, which is an ordered sequence of all directory entries in a particular directory.

Issue 4, Version 2

The *ERRORS* section is updated for X/OPEN UNIX conformance as follows:

- It states that [ELOOP] will be returned if too many symbolic links are encountered during pathname resolution.
- A second [ENAMETOOLONG] condition is defined that may report excessive length of an intermediate result of pathname resolution of a symbolic link.
NAME
openlog — open a connection to the logging facility

SYNOPSIS
#include <syslog.h>

void openlog(const char *ident, int logopt, int facility);

DESCRIPTION
Refer to closelog().

CHANGE HISTORY
First released in Issue 4, Version 2.
optarg

NAME
    optarg, opterr, optind, optopt — options parsing variables

SYNOPSIS
    #include <stdio.h>
    extern char *optarg;
    extern int opterr, optind, optopt;

DESCRIPTION
    Refer to getopt().

CHANGE HISTORY
    First released in Issue 1.
    Originally derived from Issue 1 of the SVID.

Issue 4
    Entry derived from getopt() in Issue 3, with the following change:
        • Item optopt is added to the list of external data items.
NAME

fpathconf, pathconf — get configurable pathname variables

SYNOPSIS

#include <unistd.h>

long int fpathconf(int fildes, int name);
long int pathconf(const char * path, int name);

DESCRIPTION

The *fpathconf()* and *pathconf()* functions provide a method for the application to determine the
current value of a configurable limit or option (variable) that is associated with a file or directory.

For *pathconf()* , the *path* argument points to the pathname of a file or directory.

For *fpathconf()* , the *fildes* argument is an open file descriptor.

The *name* argument represents the variable to be queried relative to that file or directory. Implementations will support all of the variables listed in the following table and may support others. The variables in the following table come from *<limits.h>* or *<unistd.h>* and the symbolic constants, defined in *<unistd.h>* , are the corresponding values used for *name* :

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value of name</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK_MAX</td>
<td>_PC_LINK_MAX</td>
<td>1</td>
</tr>
<tr>
<td>MAX_CANON</td>
<td>_PC_MAX_CANON</td>
<td>2</td>
</tr>
<tr>
<td>MAX_INPUT</td>
<td>_PC_MAX_INPUT</td>
<td>2</td>
</tr>
<tr>
<td>NAME_MAX</td>
<td>_PC_NAME_MAX</td>
<td>3,4</td>
</tr>
<tr>
<td>PATH_MAX</td>
<td>_PC_PATH_MAX</td>
<td>4,5</td>
</tr>
<tr>
<td>PIPE_BUF</td>
<td>_PC_PIPE_BUF</td>
<td>6</td>
</tr>
<tr>
<td>_POSIX_CHOWN_RESTRICTED</td>
<td>_PC_CHOWN_RESTRICTED</td>
<td>7</td>
</tr>
<tr>
<td>_POSIX_NO_TRUNC</td>
<td>_PC_NO_TRUNC</td>
<td>3,4</td>
</tr>
<tr>
<td>_POSIX_VDISABLE</td>
<td>_PC_VDISABLE</td>
<td>2</td>
</tr>
</tbody>
</table>

Notes:

1. If *path* or *fildes* refers to a directory, the value returned applies to the directory itself.
2. If *path* or *fildes* does not refer to a terminal file, it is unspecified whether an implementation supports an association of the variable name with the specified file.
3. If *path* or *fildes* refers to a directory, the value returned applies to filenames within the directory.
4. If *path* or *fildes* does not refer to a directory, it is unspecified whether an implementation supports an association of the variable name with the specified file.
5. If *path* or *fildes* refers to a directory, the value returned is the maximum length of a relative pathname when the specified directory is the working directory.
6. If *path* refers to a FIFO, or *fildes* refers to a pipe or FIFO, the value returned applies to the referenced object. If *path* or *fildes* refers to a directory, the value returned applies to any FIFO that exists or can be created within the directory. If *path* or *fildes* refers to any other type of file, it is unspecified whether an implementation supports an association of the variable name with the specified file.
7. If *path* or *fildes* refers to a directory, the value returned applies to any files, other than directories, that exist or can be created within the directory.

**RETURN VALUE**

If *name* is an invalid value, both *pathconf()* and *fpathconf()* return −1 and *errno* is set to indicate the error.

If the variable corresponding to *name* has no limit for the *path* or file descriptor, both *pathconf()* and *fpathconf()* return −1 without changing *errno*. If the implementation needs to use *path* to determine the value of *name* and the implementation does not support the association of *name* with the file specified by *path*, or if the process did not have appropriate privileges to query the file specified by *path*, or *path* does not exist, *pathconf()* returns −1 and *errno* is set to indicate the error.

If the implementation needs to use *fildes* to determine the value of *name* and the implementation does not support the association of *name* with the file specified by *fildes*, or if *fildes* is an invalid file descriptor, *fpathconf()* will return −1 and *errno* is set to indicate the error.

Otherwise *pathconf()* or *fpathconf()* returns the current variable value for the file or directory without changing *errno*. The value returned will not be more restrictive than the corresponding value available to the application when it was compiled with the implementation’s `<limits.h>` or `<unistd.h>`.

**ERRORS**

The *pathconf()* function will fail if:

- [EINVAL] The value of *name* is not valid.
- [ELOOP] Too many symbolic links were encountered in resolving *path*.

The *pathconf()* function may fail if:

- [EACCES] Search permission is denied for a component of the path prefix.
- [EINVAL] The implementation does not support an association of the variable *name* with the specified file.

- [ENAMETOOLONG] The length of the *path* argument exceeds [PATH_MAX] or a pathname component is longer than [NAME_MAX].
- [ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds [PATH_MAX].
- [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- [ENOTDIR] A component of the path prefix is not a directory.

The *fpathconf()* function will fail if:

- [EINVAL] The value of *name* is not valid.

The *fpathconf()* function may fail if:

- [EBADF] The *fildes* argument is not a valid file descriptor.
- [EINVAL] The implementation does not support an association of the variable *name* with the specified file.

**SEE ALSO**

*sysconf()*, `<limits.h>`, `<unistd.h>`. 

430 X/Open CAE Specification (1994)
CHANGE HISTORY
First released in Issue 3.
Entry included for alignment with the POSIX.1-1988 standard.

Issue 4
The following changes have been made for alignment with the ISO POSIX-1 standard:

- The type of argument `path` is changed from `char *` to `const char *`. Also the return value of both functions is changed from `long` to `long int`.
- In the DESCRIPTION section, the words “The behaviour is undefined if” have been replaced by “it is unspecified whether an implementation supports an association of the variable name with the specified file” in notes 2, 4, and 6.
- In the RETURN VALUE section, errors associated with the use of `path` and `fildes`, when an implementation does not support the requested association, are now specified separately.
- The requirement that `errno` be set to indicate the error is added.

The following change is incorporated for alignment with the FIPS requirements:

- In the ERRORS section, the condition whereby `[ENAMETOOLONG]` will be returned if a pathname component is larger than `{NAME_MAX}` is now defined as mandatory and marked as an extension.

Issue 4, Version 2
The ERRORS section is updated for X/Open UNIX conformance as follows:

- It states that `[ELOOP]` will be returned if too many symbolic links are encountered during pathname resolution.
- A second `[ENAMETOOLONG]` condition is defined that may report excessive length of an intermediate result of pathname resolution of a symbolic link.
pause( )

BASE

System Interfaces

NAME
pause — suspend process until signal is received

SYNOPSIS
#include <unistd.h>
int pause(void);

DESCRIPTION
The pause( ) function suspends the calling process until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process.

If the action is to terminate the process, pause() will not return.

If the action is to execute a signal-catching function, pause() will return after the signal-catching function returns.

RETURN VALUE
Since pause() suspends process execution indefinitely unless interrupted by a signal, there is no successful completion return value. A value of −1 is returned and errno is set to indicate the error.

ERRORS
The pause() function will fail if:

[EINTR] A signal is caught by the calling process and control is returned from the signal-catching function.

SEE ALSO
sigsuspend(), <unistd.h>.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:

• The argument list is explicitly defined as void.

Other changes are incorporated as follows:

• The header <unistd.h> is added to the SYNOPSIS section.

• In the RETURN VALUE section, the text is expanded to indicate that process execution is suspended indefinitely “unless interrupted by a signal”.
NAME
   pclose — close a pipe stream to or from a process

SYNOPSIS
   #include <stdio.h>
   int pclose(FILE * stream);

DESCRIPTION
   The pclose() function closes a stream that was opened by popen(), waits for the command to terminate, and returns the termination status of the process that was running the command language interpreter. However, if a call caused the termination status to be unavailable to pclose(), then pclose() returns −1 with errno set to [ECHILD] to report this situation; this can happen if the application calls one of the following functions:
      • wait()
      • waitpid() with a pid argument less than or equal to 0 or equal to the process ID of the command line interpreter
      • any other function not defined in this document that could do one of the above.
   In any case, pclose() will not return before the child process created by popen() has terminated.
   If the command language interpreter cannot be executed, the child termination status returned by pclose() will be as if the command language interpreter terminated using exit(127) or _exit(127).
   The pclose() function will not affect the termination status of any child of the calling process other than the one created by popen() for the associated stream.
   If the argument stream to pclose() is not a pointer to a stream created by popen(), the result of pclose() is undefined.

RETURN VALUE
   Upon successful return, pclose() returns the termination status of the command language interpreter. Otherwise, pclose() returns −1 and sets errno to indicate the error.

ERRORS
   The pclose() function will fail if:
      [ECHILD] The status of the child process could not be obtained, as described above.

SEE ALSO
   fork(), popen(), waitpid(), <stdio.h>.

CHANGE HISTORY
   First released in Issue 1.
   Derived from Issue 1 of the SVID.

Issue 4
   The following changes are incorporated for alignment with the ISO POSIX-2 standard:
      • The interface is no longer marked as an extension.
      • The simple DESCRIPTION section given in Issue 3 is replaced with a more complete description in this issue. In particular, interactions between this function and wait() and waitpid() are defined.
NAME
perror — write error messages to standard error

SYNOPSIS
#include <stdio.h>

void perror(const char * s);

DESCRIPTION
The perror() function maps the error number in the external variable errno
to a language-dependent error message, which is written to the standard error stream
as follows: first (if s is not a null pointer and the character pointed to by s is not
the null byte), the string pointed to by s followed by a colon and a space character;
then an error message string followed by a newline character. The contents of the
error message strings are the same as those returned by strerror() with argument
errno.

The perror() function will mark the file associated with the standard error stream
as having been written (st_ctime, st_mtime marked for update) at some time
between its successful completion and exit(), abort(), or the completion of
fflush() or fclose() on stderr.

RETURN VALUE
The perror() function returns no value.

ERRORS
No errors are defined.

SEE ALSO
strerror(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:
• A paragraph is added to the DESCRIPTION defining the effects of this function
  on the st_ctime and st_mtime fields of the standard error stream.

The following change is incorporated for alignment with the ISO C standard:
• The type of argument s is changed from char * to const char *.

Another change is incorporated as follows:
• The language for error message strings was given as implementation-dependent
  in Issue 3. In this issue, they are defined as language-dependent.
NAME

pipe — create an interprocess channel

SYNOPSIS

#include <unistd.h>
int pipe(int fildes[2]);

DESCRIPTION

The pipe() function will create a pipe and place two file descriptors, one each into the arguments fildes[0] and fildes[1], that refer to the open file descriptions for the read and write ends of the pipe. Their integer values will be the two lowest available at the time of the pipe() call. (The fcntl() function can be used to set both these flags.)

Data can be written to the file descriptor fildes[1] and read from file descriptor fildes[0]. A read on the file descriptor fildes[0] will access data written to file descriptor fildes[1] on a first-in-first-out basis. It is unspecified whether fildes[0] is also open for writing and whether fildes[1] is also open for reading.

A process has the pipe open for reading (correspondingly writing) if it has a file descriptor open that refers to the read end, fildes[0] (write end, fildes[1]).

Upon successful completion, pipe() will mark for update the st_atime, st_ctime and st_mtime fields of the pipe.

RETURN VALUE

Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS

The pipe() function will fail if:

[EMFILE] More than |OPEN_MAX| minus two file descriptors are already in use by this process.

[ENFILE] The number of simultaneously open files in the system would exceed a system-imposed limit.

SEE ALSO

fcntl(), read(), write(), <fcntl.h>, <unistd.h>.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following change is incorporated in this issue:

• The header <unistd.h> is added to the SYNOPSIS section.

Issue 4, Version 2

The DESCRIPTION is updated for X/OPEN UNIX conformance to indicate that certain dispositions of fildes[0] and fildes[1] are unspecified.
NAME
poll — input/output multiplexing

SYNOPSIS
#include <poll.h>

int poll(struct pollfd fds[], nfds_t nfds, int timeout);

DESCRIPTION
The poll() function provides applications with a mechanism for multiplexing input/output over a set of file descriptors. For each member of the array pointed to by fds, poll() examines the given file descriptor for the event(s) specified in events. The number of pollfd structures in the fds array is specified by nfds. The poll() function identifies those file descriptors on which an application can read or write data, or on which certain events have occurred.

The fds argument specifies the file descriptors to be examined and the events of interest for each file descriptor. It is a pointer to an array with one member for each open file descriptor of interest. The array's members are pollfd structures within which fd specifies an open file descriptor and events and revents are bitmasks constructed by OR-ing a combination of the following event flags:

POLLIN Data other than high-priority data may be read without blocking. For STREAMS, this flag is set in revents even if the message is of zero length.

POLLRDNORM Normal data (priority band equals 0) may be read without blocking. For STREAMS, this flag is set in revents even if the message is of zero length.

POLLRDBAND Data from a non-zero priority band may be read without blocking. For STREAMS, this flag is set in revents even if the message is of zero length.

POLLPRI High-priority data may be received without blocking. For STREAMS, this flag is set in revents even if the message is of zero length.

POLLOUT Normal data (priority band equals 0) may be written without blocking.

POLLWRNORM Same as POLLOUT.

POLLWRBAND Priority data (priority band greater than 0) may be written.

POLLERR An error has occurred on the device or stream. This flag is only valid in the revents bitmask; it is ignored in the events member.

POLLHUP The device has been disconnected. This event and POLLOUT are mutually exclusive; a stream can never be writable if a hangup has occurred. However, this event and POLLIN, POLLRDNORM, POLLRDBAND or POLLPRI are not mutually exclusive. This flag is only valid in the revents bitmask; it is ignored in the events member.

POLLNVAL The specified fd value is invalid. This flag is only valid in the revents member; it is ignored in the events member.

If the value of fd is less than 0, events is ignored and revents is set to 0 in that entry on return from poll().

In each pollfd structure, poll() clears the revents member except that where the application requested a report on a condition by setting one of the bits of events listed above, poll() sets the corresponding bit in revents if the requested condition is true. In addition, poll() sets the POLLHUP, POLLERR, and POLLNVAL flag in revents if the condition is true, even if the application did not set the corresponding bit in events.
If none of the defined events have occurred on any selected file descriptor, `poll()` waits at least `timeout` milliseconds for an event to occur on any of the selected file descriptors. If the value of `timeout` is 0, `poll()` returns immediately. If the value of `timeout` is −1, `poll()` blocks until a requested event occurs or until the call is interrupted.

Implementations may place limitations on the granularity of timeout intervals. If the requested timeout interval requires a finer granularity than the implementation supports, the actual timeout interval will be rounded up to the next supported value.

The `poll()` function is not affected by the O_NONBLOCK flag.

The `poll()` function supports regular files, terminal and pseudo-terminal devices, STREAMS-based files, FIFOs and pipes. The behaviour of `poll()` on elements of `fds` that refer to other types of file is unspecified.

Regular files always poll TRUE for reading and writing.

**RETURN VALUE**

Upon successful completion, `poll()` returns a non-negative value. A positive value indicates the total number of file descriptors that have been selected (that is, file descriptors for which the `revents` member is non-zero). A value of 0 indicates that the call timed out and no file descriptors have been selected. Upon failure, `poll()` returns −1 and sets `errno` to indicate the error.

**ERRORS**

The `poll()` function will fail if:

- [EAGAIN] The allocation of internal data structures failed but a subsequent request may succeed.
- [EINTR] A signal was caught during `poll()`.
- [EINVAL] The `nfds` argument is greater than `{OPEN_MAX}`, or one of the `fd` members refers to a STREAM or multiplexer that is linked (directly or indirectly) downstream from a multiplexer.

**SEE ALSO**

`getmsg()`, `putmsg()`, `read()`, `select()`, `write()`, `<poll.h>`, `<stropts.h>`, Section 2.5 on page 35.

**CHANGE HISTORY**

First released in Issue 4, Version 2.
NAME
popen — initiate pipe streams to or from a process

SYNOPSIS
#include <stdio.h>
FILE *popen(const char *command, const char *mode);

DESCRIPTION
The `popen()` function executes the command specified by the string `command`. It creates a pipe between the calling program and the executed command, and returns a pointer to a stream that can be used to either read from or write to the pipe.

If the implementation supports the referenced XCU specification, the environment of the executed command will be as if a child process were created within the `popen()` call using `fork()`, and the child invoked the `sh` utility using the call:

```
execl(shell path, "sh", "-c", command, (char *)0);
```

where `shell path` is an unspecified pathname for the `sh` utility.

The `popen()` function ensures that any streams from previous `popen()` calls that remain open in the parent process are closed in the new child process.

The `mode` argument to `popen()` is a string that specifies I/O mode:

1. If `mode` is `r`, when the child process is started its file descriptor `STDOUT_FILENO` will be the writable end of the pipe, and the file descriptor `fileno(stream)` in the calling process, where `stream` is the stream pointer returned by `popen()`, will be the readable end of the pipe.
2. If `mode` is `w`, when the child process is started its file descriptor `STDIN_FILENO` will be the readable end of the pipe, and the file descriptor `fileno(stream)` in the calling process, where `stream` is the stream pointer returned by `popen()`, will be the writable end of the pipe.
3. If `mode` is any other value, the result is undefined.

After `popen()`, both the parent and the child process will be capable of executing independently before either terminates.

RETURN VALUE
On successful completion, `popen()` returns a pointer to an open stream that can be used to read or write to the pipe. Otherwise, it returns a null pointer and may set `errno` to indicate the error.

ERRORS
The `popen()` function may fail if:

- `[EMFILE]` [FOPEN_MAX] streams are currently open in the calling process.
- `[EMFILE]` [STREAM_MAX] streams are currently open in the calling process.
- `[EINVAL]` The `mode` argument is invalid.

The `popen()` function may also set `errno` values as described by `fork()` or `pipe()`.
APPLICATION USAGE

Because open files are shared, a mode r command can be used as an input filter and a mode w command as an output filter.

Buffered reading before opening an input filter may leave the standard input of that filter mispositioned. Similar problems with an output filter may be prevented by careful buffer flushing, for example, with `fflush()`.

A stream opened by `popen()` should be closed by `pclose()`.

The behaviour of `popen()` is specified for values of mode of r and w. Other modes such as rb and wb might be supported by specific implementations, but these would not be portable features. Note that historical implementations of `popen()` only check to see if the first character of mode is r. Thus, a mode of `robert the robot` would be treated as mode r, and a mode of anything else would be treated as mode w.

If the application calls `waitpid()` with a pid argument greater than 0, and it still has a stream that was caled with `popen()` open, it must ensure that pid does not refer to the process started by `popen()`.

To determine whether or not the XCU specification environment is present, use the function call:

```
sysconf(_SC_2_VERSION)
```

(see `sysconf()`).

SEE ALSO

`sh`, `pclose()`, `pipe()`, `sysconf()`, `system()`, `<stdio.h>`.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated for alignment with the ISO POSIX-2 standard:

- The interface is no longer marked as an extension.
- The type of arguments `command` and `mode` are changed from `char *` to `const char *`.
- The DESCRIPTION section is completely rewritten for alignment with the ISO POSIX-2 standard, although it describes essentially the same functionality as Issue 3.
- The XCU specification’s sh utility is no longer required in all circumstances.
- The ERRORS section is added.

Another change is incorporated as follows:

- The APPLICATION USAGE section is extended. Only notes about buffer flushing are retained from Issue 3.
NAME
pow — power function

SYNOPSIS

```c
#include <math.h>

double pow(double x, double y);
```

DESCRIPTION
The `pow()` function computes the value of \( x \) raised to the power \( y, x^y \). If \( x \) is negative, \( y \) must be an integer value.

RETURN VALUE
Upon successful completion, `pow()` returns the value of \( x \) raised to the power \( y \).

If \( x \) is 0 and \( y \) is 0, 1.0 is returned.

EX
If \( y \) is NaN, or \( y \) is non-zero and \( x \) is NaN, NaN is returned and `errno` may be set to [EDOM]. If \( y \) is 0.0 and \( x \) is NaN, either 1.0 is returned, or NaN is returned and `errno` may be set to [EDOM].

EX
If \( x \) is 0.0 and \( y \) is negative, –HUGE_VAL is returned and `errno` may be set to [EDOM] or [ERANGE].

If the correct value would cause overflow, ±HUGE_VAL is returned, and `errno` is set to [ERANGE].

If the correct value would cause underflow, 0 is returned and `errno` may be set to [ERANGE].

ERRORS
The `pow()` function will fail if:

- [EDOM] The value of \( x \) is negative and \( y \) is non-integral.
- [ERANGE] The value to be returned would have caused overflow.

The `pow()` function may fail if:

EX
- [EDOM] The value of \( x \) is 0.0 and \( y \) is negative, or \( y \) is NaN.
- [ERANGE] The correct value would cause underflow.

EX
No other errors will occur.

APPLICATION USAGE
An application wishing to check for error situations should set `errno` to 0 before calling `pow()`. If `errno` is non-zero on return, or the return value is NaN, an error has occurred.

SEE ALSO
`exp()`, `isnan()`, `<math.h>`.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.
Issue 4

The following changes are incorporated in this issue:

- References to `matherr()` are removed.
- The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the ISO C standard and to rationalise error handling in the mathematics functions.
- The return value specified for [EDOM] is marked as an extension.
printf()  

NAME  
printf — print formatted output  

SYNOPSIS  
#include <stdio.h>  
int printf(const char *format, ...);  

DESCRIPTION  
Refer to fprintf().  

CHANGE HISTORY  
First released in Issue 1.  
Derived from Issue 1 of the SVID.  

Issue 4  
The following change is incorporated for alignment with the ISO C standard:  
• The type of the argument format is changed from char * to const char *.  
Another change is incorporated as follows:  
• The detailed description, including the printf() CHANGE HISTORY section is located under fprintf().
NAME
ptsname — get name of the slave pseudo-terminal device

SYNOPSIS
UX
```c
#include <stdlib.h>
char *ptsname(int fildes);
```

DESCRIPTION
The `ptsname()` function returns the name of the slave pseudo-terminal device associated with a master pseudo-terminal device. The `fildes` argument is a file descriptor that refers to the master device. The `ptsname()` function returns a pointer to a string containing the pathname of the corresponding slave device.

RETURN VALUE
Upon successful completion, `ptsname()` returns a pointer to a string which is the name of the pseudo-terminal slave device. Upon failure, `ptsname()` returns a null pointer. This could occur if `fildes` is an invalid file descriptor or if the slave device name does not exist in the file system.

ERRORS
No errors are defined.

APPLICATION USAGE
The value returned may point to a static data area that is overwritten by each call to `ptsname()`.

SEE ALSO
`grantpt()`, `open()`, `ttyname()`, `unlockpt()`, `<stdlib.h>`.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
putc — put byte on a stream

SYNOPSIS
#include <stdio.h>

int putc(int c, FILE *stream);

DESCRIPTION
The putc() function is equivalent to fputc(), except that if it is implemented as a macro it may evaluate stream more than once, so the argument should never be an expression with side-effects.

RETURN VALUE
Refer to fputc().

ERRORS
Refer to fputc().

APPLICATION USAGE
Because it may be implemented as a macro, putc() may treat a stream argument with side-effects incorrectly. In particular, putc(c, *f++) may not work correctly. Therefore, use of this function is not recommended in such situations; fputc() should be used instead.

SEE ALSO
fputc(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:

• The c argument is not allowed to be evaluated more than once.

Another change is incorporated as follows:

• The APPLICATION USAGE section now states that the use of this function is not recommended with a stream argument.
NAME
putchar — put byte on stdout stream

SYNOPSIS
#include <stdio.h>
int putchar(int c);

DESCRIPTION
The function call putchar(c) is equivalent to putc(c, stdout).

RETURN VALUE
Refer to fputc().

SEE ALSO
putc(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.
NAME
putenv — change or add value to environment

SYNOPSIS
EX
#include <stdlib.h>

int putenv(const char *string);

DESCRIPTION
The putenv() function uses the string argument to set environment variable values. The string argument should point to a string of the form "name=value". The putenv() function makes the value of the environment variable name equal to value by altering an existing variable or creating a new one. In either case, the string pointed to by string becomes part of the environment, so altering the string will change the environment. The space used by string is no longer used once a new string-defining name is passed to putenv().

RETURN VALUE
Upon successful completion, putenv() returns 0. Otherwise, it returns a non-zero value and sets errno to indicate the error.

ERRORS
The putenv() function may fail if:
[ENOMEM] Insufficient memory was available.

APPLICATION USAGE
The putenv() function manipulates the environment pointed to by environ, and can be used in conjunction with getenv().

This routine may use malloc() to enlarge the environment.

A potential error is to call putenv() with an automatic variable as the argument, then return from the calling function while string is still part of the environment.

Although string is currently defined as const char *, using a constant string as the argument is not recommended. The environment pointed to by environ has historically been classified as modifiable storage.

FUTURE DIRECTIONS
In a future revision of this document, the type of string will be changed to char *.

SEE ALSO
exec, getenv(), malloc(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:
• The header <stdlib.h> is added to the SYNOPSIS section.
• The type of argument string is changed from char * to const char *.
NAME
putmsg, putpmsg — send a message on a STREAM

SYNOPSIS

#include <stropts.h>

int putmsg(int fildes, const struct strbuf *ctlptr,
const struct strbuf *dataptr, int flags);

int putpmsg(int fildes, const struct strbuf *ctlptr,
const struct strbuf *dataptr, int band, int flags);

DESCRIPTION

The putmsg() function creates a message from a process buffer(s) and sends the message to a
STREAMS file. The message may contain either a data part, a control part, or both. The data
and control parts are distinguished by placement in separate buffers, as described below. The
semantics of each part is defined by the STREAMS module that receives the message.

The putpmsg() function does the same thing as putmsg(), but the process can send messages in
different priority bands. Except where noted, all requirements on putmsg() also pertain to
putpmsg().

The fildes argument specifies a file descriptor referencing an open STREAM. The ctlptr and
dataptr arguments each point to a strbuf structure.

The ctlptr argument points to the structure describing the control part, if any, to be included in
the message. The buf member in the strbuf structure points to the buffer where the control
information resides, and the len member indicates the number of bytes to be sent. The maxlen
member is not used by putmsg(). In a similar manner, the argument dataptr specifies the data, if
any, to be included in the message. The flags argument indicates what type of message should
be sent and is described further below.

To send the data part of a message, dataptr must not be a null pointer and the len member of
dataptr must be 0 or greater. To send the control part of a message, the corresponding values
must be set for ctlptr. No data (control) part will be sent if either

The STREAM head guarantees that the control part of a message generated by putmsg() is at
least 64 bytes in length.

For putmsg(), if a control part is specified and flags is set to RS_HIPRI, a high priority message is
sent. If no control part is specified, and flags is set to RS_HIPRI, putmsg() fails and sets errno to
[EINVAL]. If flags is set to 0, a normal message (priority band equal to 0) is sent. If a control part
and data part are not specified and flags is set to 0, no message is sent and 0 is returned.

The putmsg() function blocks if the STREAM write queue is full due to internal flow control
conditions, with the following exceptions:

- For high-priority messages, putmsg() does not block on this condition and continues
  processing the message.
For other messages, `putmsg()` does not block but fails when the write queue is full and `O_NONBLOCK` is set.

The `putmsg()` function also blocks, unless prevented by lack of internal resources, while waiting for the availability of message blocks in the STREAM, regardless of priority or whether `O_NONBLOCK` has been specified. No partial message is sent.

**RETURN VALUE**

Upon successful completion, `putmsg()` and `putpmsg()` return 0. Otherwise, they return −1 and set `errno` to indicate the error.

**ERRORS**

The `putmsg()` and `putpmsg()` functions will fail if:

- [EAGAIN] A non-priority message was specified, the `O_NONBLOCK` flag is set, and the STREAM write queue is full due to internal flow control conditions; or buffers could not be allocated for the message that was to be created.
- [EBADF] `fildes` is not a valid file descriptor open for writing.
- [EINTR] A signal was caught during `putmsg()`.
- [EINVAL] An undefined value is specified in `flags`, or `flags` is set to `RS_HIPRI` or `MSG_HIPRI` and no control part is supplied, or the STREAM or multiplexer referenced by `fildes` is linked (directly or indirectly) downstream from a multiplexer, or `flags` is set to `MSG_HIPRI` and `band` is non-zero (for `putpmsg()` only).
- [ENOSR] Buffers could not be allocated for the message that was to be created due to insufficient STREAMS memory resources.
- [ENOSTR] A STREAM is not associated with `fildes`.
- [ENXIO] A hangup condition was generated downstream for the specified STREAM.
- [EPIPE] or [EIO] The `fildes` argument refers to a STREAMS-based pipe and the other end of the pipe is closed. A SIGPIPE signal is generated for the calling process.
- [ERANGE] The size of the data part of the message does not fall within the range specified by the maximum and minimum packet sizes of the topmost STREAM module. This value is also returned if the control part of the message is larger than the maximum configured size of the control part of a message, or if the data part of a message is larger than the maximum configured size of the data part of a message.

In addition, `putmsg()` and `putpmsg()` will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of `errno` does not reflect the result of `putmsg()` or `putpmsg()` but reflects the prior error.

**SEE ALSO**

`getmsg()`, `poll()`, `read()`, `write()`, `<stropts.h>`, Section 2.5 on page 35.

**CHANGE HISTORY**

First released in Issue 4, Version 2.
NAME
   puts — put a string on standard output

SYNOPSIS
   #include <stdio.h>
   int puts(const char *s);

DESCRIPTION
   The puts() function writes the string pointed to by s, followed by a newline character, to the standard output stream stdout. The terminating null byte is not written.

   The st_ctime and st_mtime fields of the file will be marked for update between the successful execution of puts() and the next successful completion of a call to fflush() or fclose() on the same stream or a call to exit() or abort().

RETURN VALUE
   Upon successful completion, puts() returns a non-negative number. Otherwise it returns EOF, sets an error indicator for the stream and errno is set to indicate the error.

ERRORS
   Refer to fputc().

APPLICATION USAGE
   The puts() function appends a newline character, while fputs() does not.

SEE ALSO
   fputs(), fopen(), putc(), stdio(), <stdio.h>.

CHANGE HISTORY
   First released in Issue 1.
   Derived from Issue 1 of the SVID.

Issue 4
   The following change is incorporated for alignment with the ISO C standard:

   • The type of argument s is changed from char * to const char *.

   Another change is incorporated as follows:

   • In the DESCRIPTION section, the words “null character” are replaced by “null byte”.

System Interfaces and Headers Issue 4, Version 2
NAME
pututxline — put entry into user accounting database

SYNOPSIS
UX
#include <utmpx.h>

struct utmpx *pututxline(const struct utmpx *utmpx);

DESCRIPTION
Refer to endutxent().

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
putw — put a word on a stream

SYNOPSIS
#include <stdio.h>

int putw(int w, FILE *stream);

DESCRIPTION
The putw() function writes the word (that is, type int) w to the output stream (at the position at which the file offset, if defined, is pointing). The size of a word is the size of a type int and varies from machine to machine. The putw() function neither assumes nor causes special alignment in the file.

The st_ctime and st_mtime fields of the file will be marked for update between the successful execution of putw() and the next successful completion of a call to fflush() or fclose() on the same stream or a call to exit() or abort().

RETURN VALUE
Upon successful completion, putw() returns 0. Otherwise, a non-zero value is returned, the error indicators for the stream are set, and errno is set to indicate the error.

ERRORS
Refer to fputc().

APPLICATION USAGE
Because of possible differences in word length and byte ordering, files written using putw() are machine-dependent, and may not be readable using getw() on a different processor.

SEE ALSO
fopen(), fwrite(), getw(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.
NAME
  putwc — put wide character on a stream

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

wint_t putwc(wint_t wc, FILE *stream);

DESCRIPTION
The putwc() function is equivalent to fputwc(), except that if it is implemented as a macro it may evaluate stream more than once, so the argument should never be an expression with side-effects.

RETURN VALUE
Refer to fputwc().

ERRORS
Refer to fputwc().

APPLICATION USAGE
This interface is provided in order to align with some current implementations, and with possible future ISO standards.

Because it may be implemented as a macro, putwc() may treat a stream argument with side-effects incorrectly. In particular, putwc(wc, *f++) may not work correctly. Therefore, use of this function is not recommended; fputwc() should be used instead.

SEE ALSO
fputwc(), <stdio.h>, <wchar.h>.

CHANGE HISTORY
First released as a World-wide Portability Interface in Issue 4.
Derived from the MSE working draft.
NAME
putwchar — put wide character on stdout stream

SYNOPSIS

WP
#include <wchar.h>

wint_t putwchar(wint_t wc);

DESCRIPTION
The function call putwchar(wc) is equivalent to putwc(wc, stdout).

RETURN VALUE
Refer to fputwc().

SEE ALSO
fputwc(), putwc(), <wchar.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
qsort — sort a table of data

SYNOPSIS
#include <stdlib.h>

void qsort(void * base, size_t nel, size_t width
   int (*compar)(const void *, const void *));

DESCRIPTION
The qsort() function sorts an array of nel objects, the initial element of which is pointed to by base. The size of each object, in bytes, is specified by the width argument.

The contents of the array are sorted in ascending order according to a comparison function. The compar argument is a pointer to the comparison function, which is called with two arguments that point to the elements being compared. The function must return an integer less than, equal to, or greater than 0, if the first argument is considered respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is unspecified.

RETURN VALUE
The qsort() function returns no value.

ERRORS
No errors are defined.

APPLICATION USAGE
The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

SEE ALSO
<stdlib.h>.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:

- The arguments to compar() are formally defined in the SYNOPSIS section.
**NAME**
raise — send a signal to the executing process

**SYNOPSIS**
```c
#include <signal.h>
int raise(int sig);
```

**DESCRIPTION**
The `raise()` function sends the signal `sig` to the executing process.

**RETURN VALUE**
 Upon successful completion, 0 is returned. Otherwise, a non-zero value is returned and `errno` is set to indicate the error.

**ERRORS**
The `raise()` function will fail if:

- EINVAL The value of the `sig` argument is an invalid signal number.

**SEE ALSO**
`kill()`, `sigaction()`, `<signal.h>`, `<sys/types.h>`.

**CHANGE HISTORY**
First released in Issue 4.
Derived from the ANSI C standard.
NAME
rand — pseudo-random number generator

SYNOPSIS
#include <stdlib.h>
int rand (void);
void srand(unsigned int seed);

DESCRIPTION
The rand() function computes a sequence of pseudo-random integers in the range 0 to
\[ \text{RAND_MAX} \] with a period of at least \( 2^{32} \).

The srand() function uses the argument as a seed for a new sequence of pseudo-random
numbers to be returned by subsequent calls to rand(). If srand() is then called with the same
seed value, the sequence of pseudo-random numbers will be repeated. If rand() is called before
any calls to srand() are made, the same sequence will be generated as when srand() is first called
with a seed value of 1.

The implementation will behave as if no function defined in this document calls rand() or srand.

RETURN VALUE
The rand() function returns the next pseudo-random number in the sequence. The srand() function returns no value.

ERRORS
No errors are defined.

APPLICATION USAGE
The drand48() function provides a much more elaborate random number generator.

The following code defines a pair of functions which could be incorporated into applications
wishing to ensure that the same sequence of numbers is generated across different machines:

\[
\text{static unsigned long int next = 1;}
\]

int myrand(void) /* RAND_MAX assumed to be 32767 */
{
    next = next * 1103515245 + 12345;
    return ((unsigned int) (next/65536) % 32768);
}

void mysrand(unsigned int seed)
{
    next = seed;
}

SEE ALSO
drand48(), srand(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.
Issue 4

The following changes are incorporated for alignment with the ISO C standard:

- The argument list of `rand()` is explicitly defined as `void`.
- The argument `seed` is explicitly defined as `unsigned int`.

Other changes are incorporated as follows:

- The definition of `srand()` is added to the SYNOPSIS section.
- In the DESCRIPTION section, the text referring to the period of pseudo-random numbers is marked as an extension.
- The example in the APPLICATION USAGE section is updated (a) to use ISO C syntax, and (b) to avoid name clashes with standard functions.
random()  

NAME
random — generate pseudorandom number

SYNOPSIS
#include <stdlib.h>
long random(void);

DESCRIPTION
Refer to initstate().

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
read, readv — read from file

SYNOPSIS

#include <unistd.h>
ssize_t read(int fildes, void *buf, size_t nbyte);

#include <sys/uio.h>
ssize_t readv(int fildes, const struct iovec *iov, int iovcnt);

DESCRIPTION

The read() function attempts to read nbyte bytes from the file associated with the open file
descriptor, fildes, into the buffer pointed to by buf.

If nbyte is 0, read() will return 0 and have no other results.

On files that support seeking (for example, a regular file), the read() starts at a position in the file
given by the file offset associated with fildes. The file offset is incremented by the number of
bytes actually read.

Files that do not support seeking, for example, terminals, always read from the current position.
The value of a file offset associated with such a file is undefined.

No data transfer will occur past the current end-of-file. If the starting position is at or after the
end-of-file, 0 will be returned. If the file refers to a device special file, the result of subsequent
read() requests is implementation-dependent.

If the value of nbyte is greater than {SSIZE_MAX}, the result is implementation-dependent.

When attempting to read from an empty pipe or FIFO:

• If no process has the pipe open for writing, read() will return 0 to indicate end-of-file.

• If some process has the pipe open for writing and O_NONBLOCK is set, read() will return −1
  and set errno to [EAGAIN].

• If some process has the pipe open for writing and O_NONBLOCK is clear, read() will block
  until some data is written or the pipe is closed by all processes that had the pipe open for
  writing.

When attempting to read a file (other than a pipe or FIFO) that supports non-blocking reads and
has no data currently available:

• If O_NONBLOCK is set, read() will return a −1 and set errno to [EAGAIN].

• If O_NONBLOCK is clear, read() will block until some data becomes available.

• The use of the O_NONBLOCK flag has no effect if there is some data available.

The read() function reads data previously written to a file. If any portion of a regular file prior to
the end-of-file has not been written, read() returns bytes with value 0. For example, lseek() allows
the file offset to be set beyond the end of existing data in the file. If data is later written at
this point, subsequent reads in the gap between the previous end of data and the newly written
data will return bytes with value 0 until data is written into the gap.
Upon successful completion, where `nbyte` is greater than 0, `read()` will mark for update the `st_atime` field of the file, and return the number of bytes read. This number will never be greater than `nbyte`. The value returned may be less than `nbyte` if the number of bytes left in the file is less than `nbyte`, if the `read()` request was interrupted by a signal, or if the file is a pipe or FIFO or special file and has fewer than `nbyte` bytes immediately available for reading. For example, a `read()` from a file associated with a terminal may return one typed line of data.

If a `read()` is interrupted by a signal before it reads any data, it will return −1 with `errno` set to [EINTR].

A `read()` from a STREAMS file can read data in three different modes: byte-stream mode, message-nondiscard mode, and message-discard mode. The default is byte-stream mode. This can be changed using the `I_SRDOPT ioctl()` request, and can be tested with the `I_GRDOPT ioctl()` command. In byte-stream mode, `read()` retrieves data from the STREAM until as many bytes as were requested are transferred, or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.

In STREAMS message-nondiscard mode, `read()` retrieves data until as many bytes as were requested are transferred, or until a message boundary is reached. If `read()` does not retrieve all the data in a message, the remaining data is left on the STREAM, and can be retrieved by the next `read()` call. Message-discard mode also retrieves data until as many bytes as were requested are transferred, or a message boundary is reached. However, unread data remaining in a message after the `read()` returns is discarded, and is not available for a subsequent `read()`, `readv()` or `getmsg()` call.

How `read()` handles zero-byte STREAMS messages is determined by the current read mode setting. In byte-stream mode, `read()` accepts data until it has read `nbyte` bytes, or until there is no more data to read, or until a zero-byte message block is encountered. The `read()` function then returns the number of bytes read, and places the zero-byte message back on the STREAM to be retrieved by the next `read()`, `readv()` or `getmsg()` call. In message-nondiscard mode or message-discard mode, a zero-byte message returns 0 and the message is removed from the STREAM. When a zero-byte message is read as the first message on a STREAM, the message is removed from the STREAM and 0 is returned, regardless of the read mode.

A `read()` from a STREAMS file returns the data in the message at the front of the STREAM head read queue, regardless of the priority band of the message.

By default, STREAMs are in control-normal mode, in which a `read()` from a STREAMS file can only process messages that contain a data part but do not contain a control part. The `read()` fails if a message containing a control part is encountered at the STREAM head. This default action can be changed by placing the STREAM in either control-data mode or control-discard mode with the `I_SRDOPT ioctl()` command. In control-data mode, `read()` converts any control part to data and passes it to the application before passing any data part originally present in the same message. In control-discard mode, `read()` discards message control parts but returns to the process any data part in the message.

In addition, `read()` and `readv()` will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of `errno` does not reflect the result of `read()` or `readv()` but reflects the prior error. If a hangup occurs on the STREAM being read, `read()` continues to operate normally until the STREAM head read queue is empty. Thereafter, it returns 0.

The `readv()` function is equivalent to `read()`, but places the input data into the `iovcnt` buffers specified by the members of the `iov` array: `iov[0]`, `iov[1]`, ..., `iov[iovcnt-1]`. The `iovcnt` argument is valid if greater than 0 and less than or equal to `IOV_MAX`.

460 X/Open CAE Specification (1994)
Each `iovec` entry specifies the base address and length of an area in memory where data should be placed. The `readv()` function always fills an area completely before proceeding to the next.

Upon successful completion, `readv()` marks for update the `st_atime` field of the file.

**RETURN VALUE**

Upon successful completion, `read()` and `readv()` return a non-negative integer indicating the number of bytes actually read. Otherwise, the functions return −1 and set `errno` to indicate the error.

**ERRORS**

The `read()` and `readv()` functions will fail if:

- **[EAGAIN]** The O_NONBLOCK flag is set for the file descriptor and the process would be delayed in `read()` or `readv()`.
- **[EBADF]** The `fildes` argument is not a valid file descriptor open for reading.
- **[EBADMSG]** The file is a STREAM file that is set to control-normal mode and the message waiting to be read includes a control part.
- **[EINVAL]** The STREAM or multiplexer referenced by `fildes` is linked (directly or indirectly) downstream from a multiplexer.
- **[EIO]** A physical I/O error has occurred.
- **[ENXIO]** A request was made of a non-existent device, or the request was outside the capabilities of the device.
- **[EINVAL]** The `iovcnt` argument was less than or equal to 0, or greater than `{IOV_MAX}`.

The `readv()` function may fail if:

- **[EINVAL]** The sum of the `iov_len` values in the `iov` array overflowed an `ssize_t`.

The `readv()` function will fail if:

- **[EINVAL]** The `iovcnt` argument was less than or equal to 0, or greater than `{IOV_MAX}`.

SEE ALSO

`fcntl()`, `ioctl()`, `lseek()`, `open()`, `pipe()`, `<stropts.h>`, `<sys/uio.h>`, `<unistd.h>`, XBD specification, Chapter 9, General Terminal Interface.

**CHANGE HISTORY**

First released in Issue 1.

Derived from Issue 1 of the SVID.
The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- The type of the argument *buf* is changed from *char* to *void*, and the type of the argument *nbyte* is changed from *unsigned* to *size_t*.
- The DESCRIPTION section now states that the result is implementation-dependent if *nbyte* is greater than \{SSIZE_MAX\}. This limit was defined by the constant \{INT_MAX\} in Issue 3.

The following change is incorporated for alignment with the FIPS requirements:

- The last paragraph of the DESCRIPTION section now states that if *read()* is interrupted by a signal after it has successfully read some data, it will return the number of bytes read. In Issue 3 it was optional whether *read()* returned the number of bytes read, or whether it returned −1 with *errno* set to \{EINTR\}.

Other changes are incorporated as follows:

- The header `<unistd.h>` is added to the SYNOPSIS section.
- The DESCRIPTION section is rearranged for clarity and to align more closely with the ISO POSIX-1 standard. No functional changes are made other than as noted elsewhere in this CHANGE HISTORY section.
- In the ERRORS section in previous issues, generation of the \{EIO\} error depended on whether or not an implementation supported Job Control. This functionality is now defined as mandatory.
- The \{ENXIO\} error is marked as an extension.
- The APPLICATION USAGE section is removed.
- The description of \{EINTR\} is amended.

**Issue 4, Version 2**

The following changes are incorporated for X/OPEN UNIX conformance:

- The *readv()* function is added to the SYNOPSIS.
- The DESCRIPTION is updated to describe the reading of data from STREAMS files. An operational description of the *readv()* function is also added.
- References to the *readv()* function are added to the RETURN VALUE and ERRORS sections in appropriate places.
- The ERRORS section has been restructured to describe errors that apply generally (i.e., to both *read()* and *readv()*) and to describe those that apply to *readv()* specifically. The \{EBADMSG\}, \{EINVAL\} and \{EISDIR\} errors are also added.
NAME
readdir — read directory

SYNOPSIS
#include <sys/types.h>
#include <dirent.h>

struct dirent *readdir(DIR *dirp);

DESCRIPTION
The type DIR, which is defined in the header <dirent.h>, represents a directory stream, which is an ordered sequence of all the directory entries in a particular directory. Directory entries represent files; files may be removed from a directory or added to a directory asynchronously to the operation of readdir().

The readdir() function returns a pointer to a structure representing the directory entry at the current position in the directory stream specified by the argument dirp, and positions the directory stream at the next entry. It returns a null pointer upon reaching the end of the directory stream. The structure dirent defined by the <dirent.h> header describes a directory entry.

EX
If entries for dot or dot-dot exist, one entry will be returned for dot and one entry will be returned for dot-dot; otherwise they will not be returned.

The pointer returned by readdir() points to data which may be overwritten by another call to readdir() on the same directory stream. This data is not overwritten by another call to readdir() on a different directory stream.

If a file is removed from or added to the directory after the most recent call to opendir() or rewinddir(), whether a subsequent call to readdir() returns an entry for that file is unspecified.

The readdir() function may buffer several directory entries per actual read operation; readdir() marks for update the st_atime field of the directory each time the directory is actually read.

EX
If the entry names a symbolic link, the value of the d_ino member is unspecified.

RETURN VALUE
Upon successful completion, readdir() returns a pointer to an object of type struct dirent. When an error is encountered, a null pointer is returned and errno is set to indicate the error. When the end of the directory is encountered, a null pointer is returned and errno is not changed.

ERRORS
The readdir() function may fail if:

[EBADF] The dirp argument does not refer to an open directory stream.

[ENOENT] The current position of the directory stream is invalid.
EXAMPLES
The following sample code will search the current directory for the entry name:

```c
    dirp = opendir(".");
    while ((dp = readdir(dirp)) != NULL)
       if (strcmp(dp->d_name, name) == 0) {
           closedir(dirp);
           return FOUND;
       }
    closedir(dirp);
    return NOT_FOUND;
```

APPLICATION USAGE
The readdir() function should be used in conjunction with opendir(), closedir() and rewinddir() to examine the contents of the directory. As readdir() returns a null pointer both at the end of the directory and on error, an application wishing to check for error situations should set errno to 0, then call readdir(), then check errno and if it is non-zero, assume an error has occurred.

SEE ALSO
closedir(), lstat(), opendir(), rewinddir(), symlink(), <dirent.h>, <sys/types.h>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:

- The last paragraph of the DESCRIPTION section describing a restriction after fork() is added.

Other changes are incorporated as follows:

- The header <sys/types.h> is now marked as optional (OH); this header need not be included on XSI-conformant systems.

- In the DESCRIPTION section, the fact that XSI-conformant systems will return entries for dot and dot-dot is marked as an extension. This functionality is not specified in the ISO POSIX-1 standard.

- There is some rewording of the DESCRIPTION and RETURN VALUE sections. No functional changes are made other than as noted elsewhere in this CHANGE HISTORY section.

Issue 4, Version 2
The following changes are incorporated for X/OPEN UNIX conformance:

- A statement is added to the DESCRIPTION indicating the disposition of certain fields in struct dirent when an entry refers to a symbolic link.

- The [ENOENT] error is added to the ERRORS section as an optional error.
NAME
readlink — read the contents of a symbolic link

SYNOPSIS
UX
#include <unistd.h>
int readlink(const char *path, char *buf, size_t bufsiz);

DESCRIPTION
The `readlink()` function places the contents of the symbolic link referred to by `path` in the buffer `buf` which has size `bufsiz`. If the number of bytes in the symbolic link is less than `bufsiz`, the contents of the remainder of `buf` are unspecified.

RETURN VALUE
Upon successful completion, `readlink()` returns the count of bytes placed in the buffer. Otherwise, it returns a value of −1, leaves the buffer unchanged, and sets `errno` to indicate the error.

ERRORS
The `readlink()` function will fail if:

- [EACCES] Search permission is denied for a component of the path prefix of `path`.
- [EINVAL] The `path` argument names a file that is not a symbolic link.
- [EIO] An I/O error occurred while reading from the file system.
- [ENOENT] A component of `path` does not name an existing file or `path` is an empty string.
- [ELOOP] Too many symbolic links were encountered in resolving `path`.
- [ENAMETOOLONG] The length of `path` exceeds `{PATH_MAX}`, or a pathname component is longer than `{NAME_MAX}`.
- [ENOTDIR] A component of the path prefix is not a directory.

The `readlink()` function may fail if:

- [EACCES] Read permission is denied for the directory.

- [ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds `{PATH_MAX}`.

APPLICATION USAGE
Portable applications should not assume that the returned contents of the symbolic link are null-terminated.

SEE ALSO
`stat()`, `symlink()`, `<unistd.h>`.

CHANGE HISTORY
First released in Issue 4, Version 2.
readv() — vectored read from file

#include <sys/uio.h>

ssize_t readv(int fildes, const struct iovec *iov, int iovcnt);

Refer to read().

First released in Issue 4, Version 2.
NAME
realloc — memory reallocator

SYNOPSIS
#include <stdlib.h>

void *realloc(void * ptr , size_t size);

DESCRIPTION
The realloc() function changes the size of the memory object pointed to by ptr to the size
specified by size. The contents of the object will remain unchanged up to the lesser of the new
and old sizes. If the new size of the memory object would require movement of the object, the
space for the previous instantiation of the object is freed. If the new size is larger, the contents of
the newly allocated portion of the object are unspecified. If size is 0 and ptr is not a null pointer,
the object pointed to is freed. If the space cannot be allocated, the object remains unchanged.

If ptr is a null pointer, realloc() behaves like malloc() for the specified size.

If ptr does not match a pointer returned earlier by calloc(), malloc() or realloc() or if the space has
previously been deallocated by a call to free() or realloc(), the behaviour is undefined.

The order and contiguity of storage allocated by successive calls to realloc() is unspecified. The
pointer returned if the allocation succeeds is suitably aligned so that it may be assigned to a
pointer to any type of object and then used to access such an object in the space allocated (until
the space is explicitly freed or reallocated). Each such allocation will yield a pointer to an object
disjoint from any other object. The pointer returned points to the start (lowest byte address) of
the allocated space. If the space cannot be allocated, a null pointer is returned.

RETURN VALUE
Upon successful completion with a size not equal to 0, realloc() returns a pointer to the (possibly
moved) allocated space. If size is 0, either a null pointer or a unique pointer that can be
successfully passed to free() is returned. If there is not enough available memory, realloc()
returns a null pointer and sets errno to [ENOMEM].

ERRORS
The realloc() function will fail if:

EX [ENOMEM] Insufficient memory is available.

SEE ALSO
calloc(), free(), malloc(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated for alignment with the ISO C standard:

• The DESCRIPTION section is updated to indicate (a) that the order and contiguity of
storage allocated by successive calls to this function is unspecified, (b) that each allocation
yields a pointer to an object disjoint from any other object, and (c) that the returned pointer
points to the lowest byte address of the allocation.

• The RETURN VALUE section is updated to indicate what will be returned if size is 0.
Other changes are incorporated as follows:

- The setting of `errno` and the [ENOMEM] error are marked as extensions.
- The **APPLICATION USAGE** section is removed.
NAME
realpath — resolve pathname

SYNOPSIS
UX
#include <stdlib.h>
char *realpath(const char * file_name, char * resolved_name);

DESCRIPTION
The realpath() function derives, from the pathname pointed to by file_name, an absolute
pathname that names the same file, whose resolution does not involve ".", ".", or symbolic links.
The generated pathname is stored, up to a maximum of {PATH_MAX} bytes, in the buffer
pointed to by resolved_name.

RETURN VALUE
On successful completion, realpath() returns a pointer to the resolved name. Otherwise,
realpath() returns a null pointer and sets errno to indicate the error, and the contents of the buffer
pointed to by resolved_name are undefined.

ERRORS
The realpath() function will fail if:

[EACCES] Read or search permission was denied for a component of file_name.
[EINVAL] Either the file_name or resolved_name argument is a null pointer.
[EIO] An error occurred while reading from the file system.
[ELOOP] Too many symbolic links were encountered in resolving path.
[ENAMETOOLONG] The file_name argument is longer than {PATH_MAX} or a pathname
component is longer than {NAME_MAX}.
[ENOENT] A component of file_name does not name an existing file or file_name points to
an empty string.
[ENOTDIR] A component of the path prefix is not a directory.

The realpath() function may fail if:

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result
whose length exceeds {PATH_MAX}.
[ENOMEM] Insufficient storage space is available.

SEE ALSO
getcwd(), sysconf(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
re_comp( )

X/OPEN UNIX

System Interfaces

NAME
re_comp, re_exec — compile and execute regular expressions (TO BE WITHDRAWN)

SYNOPSIS
UX
#include <re_comp.h>

char *re_comp(const char *string);

int re_exec(const char *string);

DESCRIPTION
The re_comp( ) function converts a regular expression string (RE) into an internal form suitable
for pattern matching. The re_exec() function compares the string pointed to by the string
argument with the last regular expression passed to re_comp().

If re_comp() is called with a null pointer argument, the current regular expression remains
unchanged.

Strings passed to both re_comp() and re_exec() must be terminated by a null byte, and may
include newline characters.

The re_comp() and re_exec() functions support simple regular expressions, which are defined
below.

The following one-character REs match a single character:
1.1 An ordinary character (not one of those discussed in 1.2 below) is a one-character RE that
matches itself.

1.2 A backslash (\) followed by any special character is a one-character RE that matches the
special character itself. The special characters are:
   a. ., *, [ and \ (period, asterisk, left square bracket, and backslash, respectively), which
      are always special, except when they appear within square brackets ([]); see 1.4
      below).
   b. ^ (caret or circumflex), which is special at the beginning of an entire RE (see 3.1 and 3.2
      below), or when it immediately follows the left of a pair of square brackets ([]) (see
      1.4 below).
   c. $ (dollar symbol), which is special at the end of an entire RE (see 3.2 below).
   d. The character used to bound (delimit) an entire RE, which is special for that RE.

1.3 A period (.) is a one-character RE that matches any character except new-line.

1.4 A non-empty string of characters enclosed in square brackets ([]) is a one-character RE that
matches any one character in that string. If, however, the first character of the string is a
circumflex (^), the one-character RE matches any character except new-line and the
remaining characters in the string. The ^ has this special meaning only if it occurs first in the
string. The minus (-) may be used to indicate a range of consecutive ASCII characters; for
example, [0-9] is equivalent to [0123456789]. The - loses this special meaning if it occurs
first (after an initial ^, if any) or last in the string. The right square bracket (]) does not
terminate such a string when it is the first character within it (after an initial ^, if any); for
example, [a-f] matches either a right square bracket (]) or one of the letters a through f
inclusive. The four characters listed in 1.2.a above stand for themselves within such a string
of characters.
The following rules may be used to construct REs from one-character REs:

2.1 A one-character RE is a RE that matches whatever the one-character RE matches.

2.2 A one-character RE followed by an asterisk (*) is a RE that matches zero or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character RE followed by \( \{m, n\} \) is a RE that matches a range of occurrences of the one-character RE. The values of \( m \) and \( n \) must be non-negative integers less than 256; \( \{m\} \) matches exactly \( m \) occurrences; \( \{m, n\} \) matches at least \( m \) occurrences; \( \{m, n\} \) matches any number of occurrences between \( m \) and \( n \) inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.

2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

2.5 A RE enclosed between the character sequences \((\) and \(\)) is a RE that matches whatever the unadorned RE matches.

2.6 The expression \( \backslash n \) matches the same string of characters as was matched by an expression enclosed between \( \backslash( \) and \( \backslash) \) earlier in the same RE. Here \( n \) is a digit; the sub-expression specified is that beginning with the \( n \)-th occurrence of \( \backslash \) counting from the left. For example, the expression \( \backslash(.*\backslash)\backslash1\backslash\$ \) matches a line consisting of two repeated appearances of the same string.

Finally, an entire RE may be constrained to match only an initial segment or final segment of a line (or both).

3.1 A circumflex (ˆ) at the beginning of an entire RE constrains that RE to match an initial segment of a line.

3.2 A dollar symbol ($) at the end of an entire RE constrains that RE to match a final segment of a line. The construction ˆentire RE$ constrains the entire RE to match the entire line.

The null RE (that is, //) is equivalent to the last RE encountered.

The behaviour of re_comp() and re_exec() in locales other than the POSIX locale is unspecified.

RETURN VALUE

The re_comp() function returns a null pointer when the string pointed to by the string argument is successfully converted. Otherwise, a pointer to an unspecified error message string is returned.

Upon successful completion, re_exec() returns 1 if string matches the last compiled regular expression. Otherwise, re_exec() returns 0 if string fails to match the last compiled regular expression, and −1 if the compiled regular expression is invalid (indicating an internal error).

ERRORS

No errors are defined.

APPLICATION USAGE

For portability to implementations conforming to earlier versions of this document, regcomp() and regexec() are preferred to these functions.

SEE ALSO

regcomp(), <re_comp.h>.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
regcmp, regex — compile and execute regular expression (TO BE WITHDRAWN)

SYNOPSIS
UX
#include <libgen.h>
char *regcmp (const char *string1 , ... /*, (char *)0 */);
char *regex (const char *re , const char *subject , ... );
extern char *__loc1;

DESCRIPTION
The regcmp() function compiles a regular expression consisting of the concatenated arguments
and returns a pointer to the compiled form. The end of arguments is indicated by a null pointer.
The malloc() function is used to create space for the compiled form. It is the process' responsibility to free unneeded space so allocated. A null pointer returned from regcmp() indicates an invalid argument.

The regex() function executes a compiled pattern against the subject string. Additional arguments of type char * must be passed to receive matched subexpressions back. If an insufficient number of arguments is passed to accept all the values that the regular expression returns, the behaviour is undefined. A global character pointer __loc1 points to the first matched character in the subject string. Both regcmp() and regex() were largely borrowed from the editor, and are defined in re_comp(), but the syntax and semantics have been changed slightly. The following are the valid symbols and their associated meanings:

[!*].
These symbols retain their meaning as defined in re_comp().

$ Matches the end of the string; \n matches a new-line.

- Used within brackets, the hyphen signifies an ASCII character range. For example,
[a-z] is equivalent to [abcd . . . xyz]. The - can represent itself only if used as the first or last character. For example, the character class expression [\-] matches the characters ] and -.

+ A regular expression followed by + means one or more times. For example, [0-9]+ is equivalent to [0-9][0-9]*.

{m} {m,} {m,u}
Integer values enclosed in { } indicate the number of times the preceding regular expression can be applied. The value m is the minimum number and u is a number, less than 256, which is the maximum. If the value of either m or u is 256 or greater, the behaviour is undefined. The syntax {m} indicates the exact number of times the regular expression can be applied. The syntax {m,} is analogous to {m,infinity}. The plus (+) and asterisk (*) operations are equivalent to {1,} and {0,} respectively.

(...)$n The value of the enclosed regular expression is returned. The value is stored in the (n+1)th argument following the subject argument. A maximum of ten enclosed regular expressions are allowed. The regex() function makes its assignments unconditionally.

(...) Parentheses are used for grouping. An operator, such as *, +, or { }, can work on a single character or a regular expression enclosed in parentheses. For example, (a*{cb+})*$0.

Since all of the above defined symbols are special characters, they must be escaped to be used as themselves.
The behaviour of regcmp() and regex() in locales other than the POSIX locale is unspecified.

RETURN VALUE
Upon successful completion, regcmp() returns a pointer to the compiled regular expression. Otherwise, a null pointer is returned and errno may be set to indicate the error.

Upon successful completion, regex() returns a pointer to the next unmatched character in the subject string. Otherwise, a null pointer is returned.

The regex() function returns a null pointer on failure, or a pointer to the next unmatched character on success.

ERRORS
The regcmp() function may fail if:
[ENOMEM] Insufficient storage space was available.

No errors are defined for regex().

APPLICATION USAGE
For portability to implementations conforming to earlier versions of this document, regcomp() is preferred over this function.

User programs that use regcmp() may run out of memory if regcmp() is called iteratively without freeing compiled regular expression strings that are no longer required.

SEE ALSO
malloc(), regcomp(), <libgen.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME

regcomp( ), regexec( ), regerror( ), regfree( ) — regular expression matching

SYNOPSIS

```
#include <sys/types.h>
#include <regex.h>

int regcomp(regex_t *preg, const char *pattern, int cflags);
int regexec(const regex_t *preg, const char *string,
            size_t nmatch, regmatch_t pmatch[], int eflags);
size_t regerror(int errcode, const regex_t *preg,
                char *errbuf, size_t errbuf_size);
void regfree(regex_t *preg);
```

DESCRIPTION

These functions interpret *basic* and *extended* regular expressions as described in the XBD specification, *Chapter 7, Regular Expressions*.

The structure type `regex_t` contains at least the following member:

<table>
<thead>
<tr>
<th>Member Type</th>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_t</td>
<td>re_nsub</td>
<td>Number of parenthesised subexpressions.</td>
</tr>
</tbody>
</table>

The structure type `regmatch_t` contains at least the following members:

<table>
<thead>
<tr>
<th>Member Type</th>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regoff_t</td>
<td>rm_so</td>
<td>Byte offset from start of <em>string</em> to start of substring.</td>
</tr>
<tr>
<td>regoff_t</td>
<td>rm_eo</td>
<td>Byte offset from start of <em>string</em> of the first character after the end of substring.</td>
</tr>
</tbody>
</table>

The `regcomp()` function will compile the regular expression contained in the string pointed to by the *pattern* argument and place the results in the structure pointed to by *preg*. The *cflags* argument is the bitwise inclusive OR of zero or more of the following flags, which are defined in the header `<regex.h>`:

- **REG_EXTENDED** Use Extended Regular Expressions.
- **REG_ICASE** Ignore case in match. (See the XBD specification, *Chapter 7, Regular Expressions.*)
- **REG_NOSUB** Report only success/fail in `regexec()`.  
- **REG_NEWLINE** Change the handling of newline characters, as described in the text.

The default regular expression type for *pattern* is a Basic Regular Expression. The application can specify Extended Regular Expressions using the REG_EXTENDED *cflags* flag.

On successful completion, it returns 0; otherwise it returns non-zero, and the content of *preg* is undefined.

If the REG_NOSUB flag was not set in *cflags*, then `regcomp()` will set `re_nsub` to the number of parenthesised subexpressions (delimited by `\(` or `\)` in basic regular expressions or `(` or `)` in extended regular expressions) found in *pattern*. 
The `regexec()` function compares the null-terminated string specified by `string` with the compiled regular expression `preg` initialised by a previous call to `regcomp()`. If it finds a match, `regexec()` returns 0; otherwise it returns non-zero indicating either no match or an error. The `eflags` argument is the bitwise inclusive OR of zero or more of the following flags, which are defined in the header `<regex.h>`:

- **REG_NOTBOL**: The first character of the string pointed to by `string` is not the beginning of the line. Therefore, the circumflex character (\^), when taken as a special character, will not match the beginning of `string`.

- **REG_NOTEOL**: The last character of the string pointed to by `string` is not the end of the line. Therefore, the dollar sign ($), when taken as a special character, will not match the end of `string`.

If `nmatch` is 0 or REG_NOSUB was set in the `cflags` argument to `regcomp()`, then `regexec()` will ignore the `pmatch` argument. Otherwise, the `pmatch` argument must point to an array with at least `nmatch` elements, and `regexec()` will fill in the elements of that array with offsets of the substrings of `string` that correspond to the parenthesised subexpressions of `pattern`. `pmatch[i].rm_so` will be the byte offset of the beginning and `pmatch[i].rm_eo` will be one greater than the byte offset of the end of substring `i`. (Subexpression `i` begins at the `i`th matched open parenthesis, counting from 1.) Offsets in `pmatch[0]` identify the substring that corresponds to the entire regular expression. Unused elements of `pmatch` up to `pmatch[nmatch–1]` will be filled with –1. If there are more than `nmatch` subexpressions in `pattern` (pattern itself counts as a subexpression), then `regexec()` will still do the match, but will record only the first `nmatch` substrings.

When matching a basic or extended regular expression, any given parenthesised subexpression might participate in the match of several different substrings of `string`, or it might not match any substring even though the pattern as a whole did match. The following rules are used to determine which substrings to report in `pmatch` when matching regular expressions:

1. If subexpression `i` in a regular expression is not contained within another subexpression, and it participated in the match several times, then the byte offsets in `pmatch[i]` will delimit the last such match.

2. If subexpression `i` is not contained within another subexpression, and it did not participate in an otherwise successful match, the byte offsets in `pmatch[i]` will be –1. A subexpression does not participate in the match when:
   - `*` or `\{` `\}` appears immediately after the subexpression in a basic regular expression, or
   - `*`, `?`, or `|` appears immediately after the subexpression in an extended regular expression, and the subexpression did not match (matched 0 times)
   - or:
     - `|` is used in an extended regular expression to select this subexpression or another, and the other subexpression matched.

3. If subexpression `i` is contained within another subexpression `j`, and `i` is not contained within any other subexpression that is contained within `j`, and a match of subexpression `j` is reported in `pmatch[j]`, then the match or non-match of subexpression `i` reported in `pmatch[i]` will be as described in 1. and 2. above, but within the substring reported in `pmatch[j]` rather than the whole string.

4. If subexpression `i` is contained in subexpression `j`, and the byte offsets in `pmatch[j]` are –1, then the pointers in `pmatch[i]` also will be –1.
5. If subexpression \( i \) matched a zero-length string, then both byte offsets in \( p\text{match}[i] \) will be
the byte offset of the character or null terminator immediately following the zero-length
string.

If, when \( \text{regexec}() \) is called, the locale is different from when the regular expression was
compiled, the result is undefined.

If \( \text{REG\_NEWLINE} \) is not set in \( \text{cflags} \), then a newline character in \( \text{pattern} \) or \( \text{string} \) will be treated
as an ordinary character. If \( \text{REG\_NEWLINE} \) is set, then newline will be treated as an ordinary
character except as follows:

1. A newline character in \( \text{string} \) will not be matched by a period outside a bracket expression
or by any form of a non-matching list (see the \textit{XBD} specification, \textit{Chapter 7, Regular
Expressions}).

2. A circumflex ('\(^{\prime}\) in \( \text{pattern} \), when used to specify expression anchoring (see the \textit{XBD}
specification, \textit{Section 7.3.8, BRE Expression Anchoring}), will match the zero-length string
immediately after a newline in \( \text{string} \), regardless of the setting of \text{REG\_NOTBOL}.

3. A dollar-sign ('$') in \( \text{pattern} \), when used to specify expression anchoring, will match the
zero-length string immediately before a newline in \( \text{string} \), regardless of the setting of
\text{REG\_NOTEOL}.

The \( \text{regfree}() \) function frees any memory allocated by \( \text{regcomp}() \) associated with \emph{preg}.

The following constants are defined as error return values:

\begin{itemize}
  \item \text{REG\_NOMATCH}   \( \text{regexec}() \) failed to match.
  \item \text{REG\_BADPAT}    Invalid regular expression.
  \item \text{REG\_ECOLLA}   Validating collating element referenced.
  \item \text{REG\_ECTYPE}   Invalid character class type referenced.
  \item \text{REG\_EESCAPE}  Trailing \textbackslash{} in \emph{pattern}.
  \item \text{REG\_ESUBREG}  Number in \textbackslash{}\textit{digit} invalid or in error.
  \item \text{REG\_EBRACK}   \[ \] imbalance.
  \item \text{REG\_ENOSYS}   The function is not supported.
  \item \text{REG\_EPAREN}   \textbackslash{}( \textbackslash{} or ( imbalance.
  \item \text{REG\_EBRACE}   \textbackslash{}\textbackslash{} imbalance.
  \item \text{REG\_BADBR}    Content of \textbackslash{}\textbackslash{} invalid: not a number, number too large, more than two
numbers, first larger than second.
  \item \text{REG\_ERANGE}   Invalid endpoint in range expression.
  \item \text{REG\_ESPACE}   Out of memory.
  \item \text{REG\_BADRPT}  ?, *, or + not preceded by valid regular expression.
\end{itemize}

The \( \text{regerror}() \) function provides a mapping from error codes returned by \( \text{regcomp}() \) and
\( \text{regexec}() \) to unspecified printable strings. It generates a string corresponding to the value of the
\textit{errcode} argument, which must be the last non-zero value returned by \( \text{regcomp}() \) or \( \text{regexec}() \) with
the given value of \emph{preg}. If \textit{errcode} is not such a value, the content of the generated string is
unspecified.

If \emph{preg} is a null pointer, but \textit{errcode} is a value returned by a previous call to \( \text{regexec}() \) or \( \text{regcomp}() \),
the \( \text{regerror}() \) still generates an error string corresponding to the value of \textit{errcode}, but it might not
be as detailed under some implementations.

If the \texttt{errbuf\_size} argument is not 0, \texttt{regerror()} will place the generated string into the buffer of size \texttt{errbuf\_size} bytes pointed to by \texttt{errbuf}. If the string (including the terminating null) cannot fit in the buffer, \texttt{regerror()} will truncate the string and null-terminate the result.

If \texttt{errbuf\_size} is 0, \texttt{regerror()} ignores the \texttt{errbuf} argument, and returns the size of the buffer needed to hold the generated string.

If the \texttt{preg} argument to \texttt{regexec()} or \texttt{regfree()} is not a compiled regular expression returned by \texttt{regcomp()}, the result is undefined. A \texttt{preg} is no longer treated as a compiled regular expression after it is given to \texttt{regfree()}.

**RETURN VALUE**

On successful completion, the \texttt{regcomp()} function returns 0. Otherwise, it returns an integer value indicating an error as described in \texttt{<regex.h>}, and the content of \texttt{preg} is undefined.

On successful completion, the \texttt{regexec()} function returns 0. Otherwise it returns \texttt{REG\_NOMATCH} to indicate no match, or \texttt{REG\_ENOSYS} to indicate that the function is not supported.

Upon successful completion, the \texttt{regerror()} function returns the number of bytes needed to hold the entire generated string. Otherwise, it returns 0 to indicate that the function is not implemented.

The \texttt{regfree()} function returns no value.

**ERRORS**

No errors are defined.

**EXAMPLES**

```c
#include <regex.h>

/*
* Match string against the extended regular expression in
* pattern, treating errors as no match.
* return 1 for match, 0 for no match
*/

int
match(const char *string, char *pattern)
{
    int status;
    regex_t re;

    if (regcomp(&re, pattern, REG\_EXTENDED | REG\_NOSUB) != 0) {
        return(0); /* report error */
    }
    status = regexec(&re, string, (size_t) 0, NULL, 0);
    regfree(&re);
    if (status != 0) {
        return(0); /* report error */
    }
    return(1);
}
```
The following demonstrates how the REG_NOTBOL flag could be used with `regexec()` to find all substrings in a line that match a pattern supplied by a user. (For simplicity of the example, very little error checking is done.)

```c
(void) regcomp (&re, pattern, 0);
/* this call to regexec() finds the first match on the line */
error = regexec (&re, &buffer[0], 1, &pm, 0);
while (error == 0) { /* while matches found */
    /* substring found between pm.rm_so and pm.rm_eo */
    /* This call to regexec() finds the next match */
    error = regexec (&re, buffer + pm.rm_eo, 1, &pm, REG_NOTBOL);
}
```

**APPLICATION USAGE**

An application could use:

```c
regerror(code,preg,(char *)NULL,(size_t)0)
```

to find out how big a buffer is needed for the generated string, `malloc()` a buffer to hold the string, and then call `regerror()` again to get the string. Alternately, it could allocate a fixed, static buffer that is big enough to hold most strings, and then use `malloc()` to allocate a larger buffer if it finds that this is too small.

To match a pattern as described in the XCU specification, Section 2.13, Pattern Matching Notation use the `fnmatch()` function.

**SEE ALSO**

`fnmatch()`, `glob()`, `<regex.h>`, `<sys/types.h>`.

**CHANGE HISTORY**

First released in Issue 4.

Derived from the ISO POSIX-2 standard.
NAME
regexp — execute regular expression (TO BE WITHDRAWN)

SYNOPSIS
UX
#include <libgen.h>
char *regexp (const char *re, const char *subject, ...);

DESCRIPTION
Refer to regexp().

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
advance, compile, step, loc1, loc2, locs — compile and match regular expressions (TO BE WITHDRAWN)

SYNOPSIS
EX
#define INIT declarations
#define GETC() getc code
#define PEEK() peek code
#define UNGETC() ungetc code
#define RETURN(ptr) return code
#define ERROR(val) error code

#include <regexp.h>
char *compile(char * instring , char * expbuf ,
              const char * endbuf , int eof );
int step(const char * string , const char * expbuf );
int advance(const char * string , const char * expbuf );
extern char *loc1, *loc2, *locs;

DESCRIPTION
These are general-purpose, regular expression-matching functions to be used in programs that
perform regular expression matching, using the Regular Expressions described in Simple
Regular Expressions (Historical Version) on page 482. These functions are defined by the
<regexp.h> header.

Implementations may also accept internationalised simple regular expressions as input.

Programs must have the following five macros declared before the #include <regexp.h>
statement. These macros are used by compile(). The macros GETC(), PEEKC() and UNGETC()
operate on the regular expression given as input to compile().

GETC() This macro returns the value of the next character (byte) in the regular
expression pattern. Successive calls to GETC() should return successive
characters of the regular expression.

PEEK() This macro returns the next character (byte) in the regular expression.
Immediately successive calls to PEEK() should return the same byte, which
should also be the next character returned by GETC().

UNGETC(c) This macro causes the argument c to be returned by the next call to GETC() and
PEEK(). No more than one character of pushback is ever needed and
this character is guaranteed to be the last character read by GETC(). The
value of the macro UNGETC(c) is always ignored.

RETURN(ptr) This macro is used on normal exit of the compile() function. The value of the
argument ptr is a pointer to the character after the last character of the
compiled regular expression. This is useful to programs that have memory
allocation to manage.

ERROR(val) This macro is the abnormal return from compile(). The argument val is an
error number (see the ERRORS section below for meanings). This call should
never return.

The step() and advance() functions do pattern matching given a character string and a compiled
regular expression as input.
The `compile()` function takes as input a simple regular expression (see Simple Regular Expressions (Historical Version) on page 482) and produces a compiled expression that can be used with `step()` and `advance()`.

The first parameter `instring` is never used explicitly by `compile()` but is useful for programs that pass down different pointers to input characters. It is sometimes used in the INIT declaration (see below). Programs which invoke functions to input characters or have characters in an external array can pass down (char*) 0 for this parameter.

The next parameter `expbuf` is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter `endbuf` is one more than the highest address where the compiled regular expression may be placed. If the compiled expression cannot fit in `(endbuf−expbuf)` bytes, a call to ERROR(50) is made.

The parameter `eof` is the character which marks the end of the regular expression.

Each program that includes the `<regexp.h>` header must have a `#define` statement for INIT. It is used for dependent declarations and initialisations. Most often it is used to set a register variable to point to the beginning of the regular expression so that this register variable can be used in the declarations for GETC(), PEEKC() and UNGETC(). Otherwise it can be used to declare external variables that might be used by GETC(), PEEKC() and UNGETC(). See the EXAMPLES section below.

The first parameter to `step()` is a pointer to a string of characters to be checked for a match. This string should be null-terminated.

The second parameter, `expbuf`, is the compiled regular expression which was obtained by a call to `compile`.

The `step()` function returns non-zero if some substring of `string` matches the regular expression in `expbuf`, and 0, if there is no match. If there is a match, two external character pointers are set as a side effect to the call to `step()`. The variable `loc1` points to the first character that matched the regular expression; the variable `loc2` points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire input string, `loc1` will point to the first character of `string` and `loc2` will point to the null at the end of `string`.

The `advance()` function returns non-zero if the initial substring of `string` matches the regular expression in `expbuf`. If there is a match an external character pointer, `loc2`, is set as a side effect.

The variable `loc2` points to the next character in `string` after the last character that matched.

When `advance()` encounters a * or \{ \} sequence in the regular expression, it will advance its pointer to the string to be matched as far as possible and will recursively call itself trying to match the rest of the string to the rest of the regular expression. As long as there is no match, `advance()` will back up along the string until it finds a match or reaches the point in the string that initially matched the * or \{ \}. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer `locs` is equal to the point in the string at some time during the backing up process, `advance()` will break out of the loop that backs up and will return 0.

The external variables `ciref`, `sed` and `nbra` are reserved.
Simple Regular Expressions (Historical Version)

A Simple Regular Expression (SRE) specifies a set of character strings. A member of this set of strings is said to be matched by the SRE.

A pattern is constructed from one or more SREs. An SRE consists of ordinary characters or metacharacters.

Within a pattern, all alphanumeric characters that are not part of a bracket expression, back-reference or duplication match themselves, that is to say, the SRE pattern abc, when applied to a set of strings, will match only those strings containing the character sequence abc anywhere in them.

Most other characters also match themselves. However, a small set of characters, known as the metacharacters, have special meanings when encountered in patterns. They are described below.

Simple Regular Expression Construction

SREs are constructed as follows:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>The character c, where c is not a special character.</td>
</tr>
<tr>
<td>\c</td>
<td>The character c, where c is any character with special meaning, see below.</td>
</tr>
<tr>
<td>^</td>
<td>The beginning of the string being compared.</td>
</tr>
<tr>
<td>$</td>
<td>The end of the string being compared.</td>
</tr>
<tr>
<td>.</td>
<td>Any character.</td>
</tr>
<tr>
<td>[s]</td>
<td>Any character in the non-empty set s, where s is a sequence of characters. Ranges may be specified as c–c. The character ] may be included in the set by placing it first in the set. The character ^ may be included in the set by placing it first or last in the set. The character * may be included in the set by placing it anywhere other than first in the set, see below. Ranges in Simple Regular Expressions are only valid if the LC_COLLATE category is set to the C locale. Otherwise, the effect of using the range notation is unspecified.</td>
</tr>
<tr>
<td>[^s]</td>
<td>Any character not in the set s, where s is defined as above.</td>
</tr>
<tr>
<td>r*</td>
<td>Zero or more successive occurrences of the regular expression r. The longest leftmost match is chosen.</td>
</tr>
<tr>
<td>rx</td>
<td>The occurrence of regular expression r followed by the occurrence of regular expression x. (Concatenation.)</td>
</tr>
<tr>
<td>r{m,n}</td>
<td>Any number of m through n successive occurrences of the regular expression r. The regular expression (r{m}) matches exactly m occurrences, (r{m,n}) matches at least m occurrences. The maximum number of occurrences is matched.</td>
</tr>
<tr>
<td>(r)</td>
<td>The regular expression r. The ( and ) sequences are ignored.</td>
</tr>
<tr>
<td>\n</td>
<td>When \n (where n is a number in the range 1 to 9) appears in a concatenated regular expression, it stands for the regular expression x, where x is the nth regular expression enclosed in ( and ) sequences that appeared earlier in the concatenated regular expression. For example, in the pattern (r) (x) (y) the \2 matches the regular expression y, giving rxyz.</td>
</tr>
</tbody>
</table>
Characters that have special meaning except where they appear within square brackets, [], or are preceded by \ are: . , * , [ , \. Other special characters, such as $ have special meaning in more restricted contexts.

The character ^ at the beginning of an expression permits a successful match only immediately after a newline or at the beginning of each of the strings to which the match is applied, and the character $ at the end of an expression requires a trailing newline.

Two characters have special meaning only when used within square brackets. The character – denotes a range, [c–c], unless it is just after the left square bracket or before the right square bracket, [–c] or [c–], in which case it has no special meaning. The character ^ has the meaning complement of if it immediately follows the left square bracket, [^c]. Elsewhere between brackets, [c^], it stands for the ordinary character ^.

The special meaning of the \ operator can be escaped only by preceding it with another \, that is, \\.

**SRE Operator Precedence**

The precedence of the operators is as shown below:

```
[... ] high precedence
* .
concatenation low precedence
```

**Internationalised SREs**

Character expressions within square brackets are constructed as follows:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>The single character c where c is not a special character.</td>
</tr>
<tr>
<td>[[:class:]]</td>
<td>A character class expression. Any character of type class, as defined by category LC_CTYPE in the program's locale (see the XBD specification, Chapter 5, Locale). For class, one of the following should be substituted:</td>
</tr>
<tr>
<td>alpha</td>
<td>a letter</td>
</tr>
<tr>
<td>upper</td>
<td>an upper-case letter</td>
</tr>
<tr>
<td>lower</td>
<td>a lower-case letter</td>
</tr>
<tr>
<td>digit</td>
<td>a decimal digit</td>
</tr>
<tr>
<td>xdigit</td>
<td>a hexadecimal digit</td>
</tr>
<tr>
<td>alnum</td>
<td>an alphanumeric (letter or digit)</td>
</tr>
<tr>
<td>space</td>
<td>a character producing white space in displayed text</td>
</tr>
<tr>
<td>punct</td>
<td>a punctuation character</td>
</tr>
<tr>
<td>print</td>
<td>a printing character</td>
</tr>
<tr>
<td>graph</td>
<td>a character with a visible representation</td>
</tr>
<tr>
<td>ctrl</td>
<td>a control character</td>
</tr>
</tbody>
</table>

| [[=c=]] | An equivalence class. Any collation element defined as having the same relative order in the current collation sequence as c. As an example, if A and a belong to the same equivalence class, then both [[=A=]b] and [[=a=]b] are equivalent to [Aab]. |
A collating symbol. Multi-character collating elements must be represented as collating symbols to distinguish them from single-character collating elements. As an example, if the string \texttt{ch} is a valid collating element, then \texttt{[[ .ch. ]]} will be treated as an element matching the same string of characters, while \texttt{ch} will be treated as a simple list of \texttt{c} and \texttt{h}. If the string is not a valid collating element in the current collating sequence definition, the symbol will be treated as an invalid expression.

Any collation element in the character expression range \texttt{c–c}, where \texttt{c} can identify a collating symbol or an equivalence class. If the character \texttt{–} appears immediately after an opening square bracket, for example, \texttt{[–c]}, or immediately prior to a closing square bracket, for example, \texttt{[c–]}, it has no special meaning.

Immediately following an opening square bracket, means the complement of, for example, \texttt{[‘c’]}. Otherwise, it has no special meaning.

Within square brackets, a \texttt{.} that is not part of a \texttt{[[ .cc. ]]} sequence, or a \texttt{:} that is not part of a \texttt{[[:class:]}} sequence, or an \texttt{=} that is not part of a \texttt{[[=c]=]}} sequence, matches itself.

**SRE Examples**

Below are examples of regular expressions:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab.d</td>
<td>\texttt{ab} any character \texttt{d}</td>
</tr>
<tr>
<td>ab.*d</td>
<td>\texttt{ab} any sequence of characters (including none) \texttt{d}</td>
</tr>
<tr>
<td>ab[xyz]d</td>
<td>\texttt{ab} one of \texttt{x y or z d}</td>
</tr>
<tr>
<td>ab[’c]d</td>
<td>\texttt{ab} anything except \texttt{c d}</td>
</tr>
<tr>
<td>`abcd$</td>
<td>a line containing only \texttt{abcd}</td>
</tr>
<tr>
<td>[a-d]</td>
<td>any one of a b c or d</td>
</tr>
</tbody>
</table>

**RETURN VALUE**

The \texttt{compile()} function uses the macro \texttt{RETURN()} on success and the macro \texttt{ERROR()} on failure, see above. The \texttt{step()} and \texttt{advance()} functions return non-zero on a successful match and 0 if there is no match.

**ERRORS**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Range endpoint too large.</td>
</tr>
<tr>
<td>16</td>
<td>Bad number.</td>
</tr>
<tr>
<td>25</td>
<td><code>digit</code> out of range.</td>
</tr>
<tr>
<td>36</td>
<td>Illegal or missing delimiter.</td>
</tr>
<tr>
<td>41</td>
<td>No remembered search string.</td>
</tr>
<tr>
<td>42</td>
<td><code>( \)</code> imbalance.</td>
</tr>
<tr>
<td>43</td>
<td>Too many <code>\</code>.</td>
</tr>
<tr>
<td>44</td>
<td>More than two numbers given in <code>\{ \}</code>.</td>
</tr>
<tr>
<td>45</td>
<td><code>)</code> expected after <code>\</code>.</td>
</tr>
<tr>
<td>46</td>
<td>First number exceeds second in <code>\{ \}</code>.</td>
</tr>
<tr>
<td>49</td>
<td><code>[ ]</code> imbalance.</td>
</tr>
<tr>
<td>50</td>
<td>Regular expression overflow.</td>
</tr>
</tbody>
</table>
EXAMPLES

The following is an example of how the regular expression macros and calls might be defined by an application program:

```
#define INIT char *sp = instring;
#define GETC( ) (*sp++)
#define PEEKC( ) (*sp)
#define UNGETC(c) (−−sp)
#define RETURN(c) return;
#define ERROR(c) regerr()

#include <regexp.h>

... (void) compile(*argv, expbuf, &expbuf[ESIZE], ´\0´);
... if (step(linebuf, expbuf) )
    succeed();
```

APPLICATION USAGE

These functions are kept for historical reasons, but will be withdrawn in a future issue of this document.

New applications should use the new functions `fnmatch()`, `glob()`, `regcomp()` and `regexec()`, which provide full internationalised regular expression functionality compatible with the ISO POSIX-2 standard, as described in the XBD specification, Chapter 7, Regular Expressions.

SEE ALSO

`fnmatch()`, `glob()`, `regcomp()`, `regexec()`, `setlocale()`, `<regex.h>`, `<regexp.h>`, the XBD specification, Chapter 7, Regular Expressions.

CHANGE HISTORY

First released in Issue 2.

Derived from Issue 2 of the SVID.

Issue 4

The following changes are incorporated in this issue:

- The interface is marked TO BE WITHDRAWN, because improved functionality is now provided by interfaces introduced for alignment with the ISO POSIX-2 standard.
- The type of the arguments `endbuf`, `string` and `expbuf` is changed from `char *` to `const char *`.
- In the DESCRIPTION section some of the text is reworded to improve clarity.
- The APPLICATION USAGE section is added.
- The example is corrected.
- The FUTURE DIRECTIONS section is removed.
remainder() — remainder function

**SYNOPSIS**

```c
#include <math.h>

double remainder(double x, double y);
```

**DESCRIPTION**

The `remainder()` function returns the floating point remainder \( r = x - ny \) when \( y \) is non-zero. The value \( n \) is the integral value nearest the exact value \( x/y \). When \( | n - x/y | = \frac{1}{2} \), the value \( n \) is chosen to be even.

The behaviour of `remainder()` is independent of the rounding mode.

**RETURN VALUE**

The `remainder()` function returns the floating point remainder \( r = x - ny \) when \( y \) is non-zero.

When \( y \) is 0, `remainder()` returns (NaN or equivalent if available) and sets `errno` to [EDOM].

If the value of \( x \) is ±Inf, `remainder()` returns NaN and sets `errno` to [EDOM].

If \( x \) or \( y \) is NaN, then the function returns NaN and `errno` may be set to [EDOM].

**ERRORS**

The `remainder()` function will fail if:

[EDOM] The \( y \) argument is 0 or the \( x \) argument is positive or negative infinity.

The `remainder()` function may fail if:

[EDOM] The \( x \) or \( y \) argument is NaN.

**SEE ALSO**

`abs()`, `<math.h>`.

**CHANGE HISTORY**

First released in Issue 4, Version 2.
NAME

remove — remove files

SYNOPSIS

```c
#include <stdio.h>

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

DESCRIPTION

The `remove()` function causes the file named by the pathname pointed to by `path` to be no longer
accessible by that name. A subsequent attempt to open that file using that name will fail, unless
it is created anew.

- If `path` does not name a directory, `remove(path)` is equivalent to `unlink(path)`.
- If `path` names a directory, `remove(path)` is equivalent to `rmdir(path)`.

RETURN VALUE

- Refer to `rmdir()` or `unlink()`.

ERRORS

- Refer to `rmdir()` or `unlink()`.

SEE ALSO

`rmdir()`, `unlink()`, `<stdio.h>`.

CHANGE HISTORY

First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard and the ISO C standard.

Issue 4

The following changes are incorporated for alignment with the ISO C standard:

- The type of argument `path` is changed from `char *` to `const char *`.
- The DESCRIPTION section is expanded to describe the operation of `remove()` more completely.

Another change is incorporated as follows:

- All statements containing references to `unlink()` and `rmdir()` in the DESCRIPTION,
  RETURN VALUE and ERRORS sections are marked as extensions.
NAME
remque — remove an element from a queue

SYNOPSIS

```c
#include <search.h>

void remque(void *element);
```

DESCRIPTION
Refer to `insque()`.

CHANGE HISTORY
First released in Issue 4, Version 2.
rename( )

NAME
rename — rename a file

SYNOPSIS
#include <stdio.h>

int rename(const char *old, const char *new);

DESCRIPTION
The rename() function changes the name of a file. The old argument points to the pathname of the file to be renamed. The new argument points to the new pathname of the file.

If the old argument and the new argument both refer to, and both link to the same existing file, rename() returns successfully and performs no other action.

If the old argument points to the pathname of a file that is not a directory, the new argument must not point to the pathname of a directory. If the link named by the new argument exists, it is removed and old renamed to new. In this case, a link named new will remain visible to other processes throughout the renaming operation and will refer either to the file referred to by new or old before the operation began. Write access permission is required for both the directory containing old and the directory containing new.

If the old argument points to the pathname of a directory, the new argument must not point to the pathname of a file that is not a directory. If the directory named by the new argument exists, it will be removed and old renamed to new. In this case, a link named new will exist throughout the renaming operation and will refer either to the file referred to by new or old before the operation began. Thus, if new names an existing directory, it must be an empty directory.

UX If old points to a pathname that names a symbolic link, the symbolic link is renamed. If new points to a pathname that names a symbolic link, the symbolic link is removed.

The new pathname must not contain a path prefix that names old. Write access permission is required for the directory containing old and the directory containing new. If the old argument points to the pathname of a directory, write access permission may be required for the directory named by old, and, if it exists, the directory named by new.

If the link named by the new argument exists and the file's link count becomes 0 when it is removed and no process has the file open, the space occupied by the file will be freed and the file will no longer be accessible. If one or more processes have the file open when the last link is removed, the link will be removed before rename() returns, but the removal of the file contents will be postponed until all references to the file are closed.

Upon successful completion, rename() will mark for update the st_ctime and st_mtime fields of the parent directory of each file.

RETURN VALUE
Upon successful completion, rename() returns 0. Otherwise, −1 is returned, errno is set to indicate the error, and neither the file named by old nor the file named by new will be changed or created.
**rename()**

**BASE**

**System Interfaces**

**ERRORS**

The `rename()` function will fail if:

- **[EACCES]** A component of either path prefix denies search permission; or one of the directories containing `old` or `new` denies write permissions; or, write permission is required and is denied for a directory pointed to by the `old` or `new` arguments.

- **[EBUSY]** The directory named by `old` or `new` is currently in use by the system or another process, and the implementation considers this an error, or the file named by `old` or `new` is a named STREAM.

- **[EEXIST] or [ENOTEMPTY]** The link named by `new` is a directory that is not an empty directory.

- **[EINVAL]** The `new` directory pathname contains a path prefix that names the `old` directory.

- **[EINVAL] or [ENOTEMPTY]** The link named by `new` is a directory that is not an empty directory.

- **[EISDIR]** The `new` argument points to a directory and the `old` argument points to a file that is not a directory.

- **[ELOOP]** Too many symbolic links were encountered in resolving either pathname.

- **[EMLINK]** The file named by `old` is a directory, and the link count of the parent directory of `new` would exceed `LINK_MAX`.

- **[ENAMETOOLONG]** The length of the `old` or `new` argument exceeds `PATH_MAX` or a pathname component is longer than `NAME_MAX`.

- **[ENOENT]** The link named by `old` does not name an existing file, or either `old` or `new` points to an empty string.

- **[ENOSPC]** The directory that would contain `new` cannot be extended.

- **[ENOTDIR]** A component of either path prefix is not a directory; or the `old` argument names a directory and `new` argument names a non-directory file.

- **[EPERM] or [EACCES]** The S_ISVTX flag is set on the directory containing the file referred to by `old` and the caller is not the file owner, nor is the caller the directory owner, nor does the caller have appropriate privileges; or `new` refers to an existing file, the S_ISVTX flag is set on the directory containing this file and the caller is not the file owner, nor is the caller the directory owner, nor does the caller have appropriate privileges.

- **[EROFS]** The requested operation requires writing in a directory on a read-only file system.

- **[EXDEV]** The links named by `new` and `old` are on different file systems and the implementation does not support links between file systems.

**UX**

**[ELOOP]** Too many symbolic links were encountered in resolving either pathname.

**FIPS**

- **[ENAMETOOLONG]** The length of the `old` or `new` argument exceeds `PATH_MAX` or a pathname component is longer than `NAME_MAX`.

The `rename()` function may fail if:

- **[ENAMETOOLONG]** Pathname resolution of a symbolic link produced an intermediate result whose length exceeds `PATH_MAX`.

490 X/Open CAE Specification (1994)
The file to be renamed is a pure procedure (shared text) file that is being executed.

SEE ALSO

`link()`, `rmdir()`, `symlink()`, `unlink()`, `<stdio.h>`.

CHANGE HISTORY

First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

**Issue 4**

The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- The type of arguments `old` and `new` are changed from `char *` to `const char *`.
- The RETURN VALUE section now states that if an error occurs, neither file will be changed or created.

The following change is incorporated for alignment with the FIPS requirements:

- In the ERRORS section, the condition whereby `ENAMETOOLONG` will be returned if a pathname component is larger than `NAME_MAX`, is now defined as mandatory and marked as an extension.

Another change is incorporated as follows:

- The `[EMLINK]` error is added to the ERRORS section.

**Issue 4, Version 2**

The following changes are made for X/OPEN UNIX conformance:

- The DESCRIPTION is updated to indicate the results of naming a symbolic link in either `old` or `new`.
- In the ERRORS section, `[EIO]` is added to indicate that a physical I/O error has occurred, `[ELOOP]` to indicate that too many symbolic links were encountered during pathname resolution, and `[EPERM]` or `[EACCES]` to indicate a permission check failure when operating on directories with `S_ISVTX` set.
- In the ERRORS section, a second `[ENAMETOOLONG]` condition is defined that may report excessive length of an intermediate result of pathname resolution of a symbolic link.
rewind()  System Interfaces

NAME
rewind — reset file position indicator in a stream

SYNOPSIS
#include <stdio.h>
void rewind(FILE *stream);

DESCRIPTION
The call:
    rewind(stream)
is equivalent to:
    (void) fseek(stream, 0L, SEEK_SET)
except that rewind() also clears the error indicator.

RETURN VALUE
The rewind() function returns no value.

ERRORS
Refer to fseek() with the exception of EINVAL which does not apply.

APPLICATION USAGE
Because rewind() does not return a value, an application wishing to detect errors should clear errno, then call rewind(), and if errno is non-zero, assume an error has occurred.

SEE ALSO
fseek(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.
NAME
rewinddir — reset position of directory stream to the beginning of a directory

SYNOPSIS
#include <sys/types.h>
#include <dirent.h>

void rewinddir(DIR *dirp);

DESCRIPTION
The rewinddir() function resets the position of the directory stream to which dirp refers to the
beginning of the directory. It also causes the directory stream to refer to the current state of the
corresponding directory, as a call to opendir() would have done. If dirp does not refer to a
directory stream, the effect is undefined.

After a call to the fork() function, either the parent or child (but not both) may continue
processing the directory stream using readdir(), rewinddir() or seekdir(). If both the parent and
child processes use these functions, the result is undefined.

RETURN VALUE
The rewinddir() function does not return a value.

ERRORS
No errors are defined.

APPLICATION USAGE
The rewinddir() function should be used in conjunction with opendir(), readdir() and closedir() to
examine the contents of the directory. This method is recommended for portability.

SEE ALSO
closedir(), opendir(), readdir(), <dirent.h>, <sys/types.h>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:

- The last paragraph of the DESCRIPTION section, describing a restriction after a fork() function is added.

Other changes are incorporated as follows:

- The header <sys/types.h> is now marked as optional (OH); this header need not be included
  on XSI-conformant systems.
rindex()  

X/OPEN UNIX  

System Interfaces

NAME
rindex — character string operations

SYNOPSIS
UX
#include <strings.h>
char *rindex(const char *s, int c);

DESCRIPTION
The rindex() function is identical to strrchr().

RETURN VALUE
See strrchr().

ERRORS
See strrchr().

APPLICATION USAGE
For portability to implementations conforming to earlier versions of this document, strrchr() is preferred over these functions.

SEE ALSO
strrchr(), <strings.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
rint — round-to-nearest integral value

SYNOPSIS
UX
#include <math.h>

double rint(double x);

DESCRIPTION
The rint() function returns the integral value (represented as a double) nearest x in the direction of the current rounding mode. The current rounding mode is implementation dependent.

If the current rounding mode rounds toward negative infinity, then rint() is identical to floor().
If the current rounding mode rounds toward positive infinity, then rint() is identical to ceil().

RETURN VALUE
Upon successful completion, the rint() function returns the integer (represented as a double precision number) nearest x in the direction of the current rounding mode.

When x is ±Inf, rint() returns x.
If the value of x is NaN, NaN is returned and errno may be set to EDOM.

ERRORS
The rint() function may fail if:
[EDOM] The x argument is NaN.

SEE ALSO
abs(), isnan(), <math.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
rmdir — remove a directory

SYNOPSIS
#include <unistd.h>
int rmdir(const char * path);

DESCRIPTION
The rmdir() function removes a directory whose name is given by path. The directory is removed only if it is an empty directory.

If the directory is the root directory or the current working directory of any process, it is unspecified whether the function succeeds, or whether it fails and sets errno to [EBUSY].

UX If path names a symbolic link, then rmdir() fails and sets errno to [ENOTDIR].

If the directory’s link count becomes 0 and no process has the directory open, the space occupied by the directory will be freed and the directory will no longer be accessible. If one or more processes have the directory open when the last link is removed, the dot and dot-dot entries, if present, are removed before rmdir() returns and no new entries may be created in the directory, but the directory is not removed until all references to the directory are closed.

Upon successful completion, the rmdir() function marks for update the st_ctime and st_mtime fields of the parent directory.

RETURN VALUE
Upon successful completion, the function rmdir() returns 0. Otherwise, −1 is returned, and errno is set to indicate the error. If −1 is returned, the named directory is not changed.

ERRORS
The rmdir() function will fail if:

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

[EBUSY] The directory to be removed is currently in use by the system or another process and the implementation considers this to be an error.

[EEXIST] or [ENOTEMPTY]
The path argument names a directory that is not an empty directory.

UX [EIO] A physical I/O error has occurred.

UX [ELOOP] Too many symbolic links were encountered in resolving path.

[ENAMETOOLONG]
The length of the path argument exceeds [PATH_MAX] or a pathname component is longer than [NAME_MAX].

FIPS [ENOENT] A component of path does not name an existing file, or the path argument names a non-existent directory or points to an empty string.

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

UX [EPERM] or [EACCES]
The S_ISVTX flag is set on the parent directory of the directory to be removed and the caller is not the owner of the directory to be removed, nor is the caller the owner of the parent directory, nor does the caller have the appropriate privileges.

496 X/Open CAE Specification (1994)
The directory entry to be removed resides on a read-only file system.

The `rmdir()` function may fail if:

- **[EROFS]** The directory entry to be removed resides on a read-only file system.

**UX**

- **[ENAMETOOLONG]** Pathname resolution of a symbolic link produced an intermediate result whose length exceeds `PATH_MAX`.

**SEE ALSO**

`mkdir()`, `remove()`, `unlink()`, `<unistd.h>`.

**CHANGE HISTORY**

First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

**Issue 4**

The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- The type of argument `path` is changed from `char *` to `const char *`.
- The `DESCRIPTION` section is expanded to indicate that, if the directory is a root directory or a current working directory, it is unspecified whether the function succeeds, or whether it fails and sets `errno` to [EBUSY]. In Issue 3, the behaviour under these circumstances was defined as “implementation-dependent”.
- The `RETURN VALUE` section is expanded to direct that if −1 is returned, the directory will not be changed.

The following change is incorporated for alignment with the FIPS requirements:

- In the `ERRORS` section, the condition whereby [ENAMETOOLONG] will be returned if a pathname component is larger that [NAME_MAX] is now defined as mandatory and marked as an extension.

Other changes are incorporated as follows:

- The header `<unistd.h>` is added to the `SYNOPSIS` section.
- The [ENAMETOOLONG] description is amended.

**Issue 4, Version 2**

The following changes are made for X/OPEN UNIX conformance:

- The `DESCRIPTION` is updated to indicate the results of naming a symbolic link in `path`.
- In the `ERRORS` section, [EIO] is added to indicate that a physical I/O error has occurred, [ELOOP] to indicate that too many symbolic links were encountered during pathname resolution, and [EPERM] or [EACCES] to indicate a permission check failure when operating on directories with S_ISVTX set.
- In the `ERRORS` section, a second [ENAMETOOLONG] condition is defined that may report excessive length of an intermediate result of pathname resolution of a symbolic link.
NAME
  sbrk — change space allocation

SYNOPSIS
  #include <unistd.h>
  
  void *sbrk(int incr);

DESCRIPTION
  Refer to brk().

CHANGE HISTORY
  First released in Issue 4, Version 2.
NAME
scalb — load exponent of a radix-independent floating-point number

SYNOPSIS
UX
#include <math.h>

double scalb(double x, double n);

DESCRIPTION
The scalb() function computes \( x \times r^n \), where \( r \) is the radix of the machine's floating point arithmetic. When \( r \) is 2, scalb() is equivalent to ldexp().

RETURN VALUE
Upon successful completion, the scalb() function returns \( x \times r^n \).
If the correct value would overflow, scalb() returns \( \pm \text{HUGE}_\text{VAL} \) (according to the sign of \( x \)) and sets errno to [ERANGE].
If the correct value would underflow, scalb() returns 0 and sets errno to [ERANGE].
The scalb() function returns \( x \) when \( x \) is \( \pm \text{Inf} \).
If \( x \) or \( n \) is NaN, then scalb() returns NaN and may set errno to [EDOM].

ERRORS
The scalb() function will fail if:
[ERANGE] The correct value would overflow or underflow.
The scalb() function may fail if:
[EDOM] The \( x \) or \( n \) argument is NaN.

APPLICATION USAGE
An application wishing to check for error situations should set errno to 0 before calling scalb(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

SEE ALSO
ldexp(), <math.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
scanf — convert formatted input

SYNOPSIS
#include <stdio.h>

int scanf(const char *format, ...);

DESCRIPTION
Refer to fscanf().

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
• The type of the argument format is changed from char * to const char *.
Other changes are incorporated as follows:
• The description of this function, including its change history, is located under fscanf().
NAME
seed48 — seed uniformly distributed pseudo-random non-negative long integer generator

SYNOPSIS
EX

```c
#include <stdlib.h>

unsigned short int *seed48(unsigned short int seed16v[3]);
```

DESCRIPTION
Refer to `drand48()`.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:
- The header `<stdlib.h>` is added to the SYNOPSIS section.
### NAME

seekdir — set position of directory stream

### SYNOPSIS

```c
#include <sys/types.h>

#include <dirent.h>

void seekdir(DIR *dirp, long int loc);
```

### DESCRIPTION

The `seekdir()` function sets the position of the next `readdir()` operation on the directory stream specified by `dirp` to the position specified by `loc`. The value of `loc` should have been returned from an earlier call to `telldir()`. The new position reverts to the one associated with the directory stream when `telldir()` was performed.

**UX** If the value of `loc` was not obtained from an earlier call to `telldir()` or if a call to `rewinddir()` occurred between the call to `telldir()` and the call to `seekdir()`, the results of subsequent calls to `readdir()` are unspecified.

### RETURN VALUE

The `seekdir()` function returns no value.

### ERRORS

No errors are defined.

### SEE ALSO

`opendir()`, `readdir()`, `telldir()`, `<dirent.h>`, `<stdio.h>`, `<sys/types.h>`.

### CHANGE HISTORY

First released in Issue 2.

**Issue 4**

The following changes are incorporated in this issue:

- The header `<sys/types.h>` is now marked as optional (OH); this header need not be included on XSI-conformant systems.
- The type of argument `loc` is expanded to `long int`.

**Issue 4, Version 2**

The DESCRIPTION is updated for X/Open UNIX conformance to indicate that a call to `readdir()` may produce unspecified results if either `loc` was not obtained by a previous call to `telldir()`, or if there is an intervening call to `rewinddir()`.
NAME

select — synchronous I/O multiplexing

SYNOPSIS

```c
#include <sys/time.h>

int select(int nfds, fd_set *readfds, fd_set *writelfds, fd_set *errorfds, struct timeval *timeout);
void FD_CLR(int fd, fd_set *fdset);
int FD_ISSET(int fd, fd_set *fdset);
void FD_SET(int fd, fd_set *fdset);
void FD_ZERO(fd_set *fdset);
```

DESCRIPTION

The `select()` function indicates which of the specified file descriptors is ready for reading, ready for writing, or has an error condition pending. If the specified condition is false for all of the specified file descriptors, `select()` blocks, up to the specified timeout interval, until the specified condition is true for at least one of the specified file descriptors.

The `select()` function supports regular files, terminal and pseudo-terminal devices, STREAMS-based files, FIFOs and pipes. The behaviour of `select()` on file descriptors that refer to other types of file is unspecified.

The `nfds` argument specifies the range of file descriptors to be tested. The `select()` function tests file descriptors in the range of 0 to `nfds`−1.

If the `readfs` argument is not a null pointer, it points to an object of type `fd_set` that on input specifies the file descriptors to be checked for being ready to read, and on output indicates which file descriptors are ready to read.

If the `writelfs` argument is not a null pointer, it points to an object of type `fd_set` that on input specifies the file descriptors to be checked for being ready to write, and on output indicates which file descriptors are ready to write.

If the `errorfds` argument is not a null pointer, it points to an object of type `fd_set` that on input specifies the file descriptors to be checked for error conditions pending, and on output indicates which file descriptors have error conditions pending.

On successful completion, the objects pointed to by the `readfs`, `writelfs`, and `errorfds` arguments are modified to indicate which file descriptors are ready for reading, ready for writing, or have an error condition pending, respectively. For each file descriptor less than `nfds`, the corresponding bit will be set on successful completion if it was set on input and the associated condition is true for that file descriptor.

If the `timeout` argument is not a null pointer, it points to an object of type `struct timeval` that specifies a maximum interval to wait for the selection to complete. If the `timeout` argument points to an object of type `struct timeval` whose members are 0, `select()` does not block. If the `timeout` argument is a null pointer, `select()` blocks until an event causes one of the masks to be returned with a valid (non-zero) value. If the time limit expires before any event occurs that would cause one of the masks to be set to a non-zero value, `select()` completes successfully and returns 0.

Implementations may place limitations on the maximum timeout interval supported. On all implementations, the maximum timeout interval supported will be at least 31 days. If the `timeout` argument specifies a timeout interval greater than the implementation-dependent maximum value, the maximum value will be used as the actual timeout value. Implementations
may also place limitations on the granularity of timeout intervals. If the requested timeout interval requires a finer granularity than the implementation supports, the actual timeout interval will be rounded up to the next supported value.

If the readfs, writefs, and errorfds arguments are all null pointers and the timeout argument is not a null pointer, select() blocks for the time specified, or until interrupted by a signal. If the readfs, writefs, and errorfds arguments are all null pointers and the timeout argument is a null pointer, select() blocks until interrupted by a signal.

File descriptors associated with regular files always select true for ready to read, ready to write, and error conditions.

On failure, the objects pointed to by the readfs, writefs, and errorfds arguments are not modified. If the timeout interval expires without the specified condition being true for any of the specified file descriptors, the objects pointed to by the readfs, writefs, and errorfds arguments have all bits set to 0.

File descriptor masks of type fd_set can be initialised and tested with FD_CLR(), FD_ISSET(), FD_SET(), and FD_ZERO(). It is unspecified whether each of these is a macro or a function. If a macro definition is suppressed in order to access an actual function, or a program defines an external identifier with any of these names, the behaviour is undefined.

FD_CLR(fd, &fdset) Clears the bit for the file descriptor fd in the file descriptor set fdset.
FD_ISSET(fd, &fdset) Returns a non-zero value if the bit for the file descriptor fd is set in the file descriptor set pointed to by fdset, and 0 otherwise.
FD_SET(fd, &fdset) Sets the bit for the file descriptor fd in the file descriptor set fdset.
FD_ZERO(&fdset) Initialises the file descriptor set fdset to have zero bits for all file descriptors.

The behaviour of these macros is undefined if the fd argument is less than 0 or greater than or equal to FD_SETSIZE.

RETURN VALUE
FD_CLR(), FD_SET() and FD_ZERO() return no value. FD_ISSET() a non-zero value if the bit for the file descriptor fd is set in the file descriptor set pointed to by fdset, and 0 otherwise.

On successful completion, select() returns the total number of bits set in the bit masks. Otherwise, −1 is returned, and errno is set to indicate the error.

ERRORS
Under the following conditions, select() fails and sets errno to:

[EBADF] One or more of the file descriptor sets specified a file descriptor that is not a valid open file descriptor.

[EINTR] The select() function was interrupted before any of the selected events occurred and before the timeout interval expired.

If SA_RESTART has been set for the interrupting signal, it is implementation-dependent whether select() restarts or returns with [EINTR].

[EINVAL] An invalid timeout interval was specified.

[EINVAL] The nfds argument is less than 0, or greater than or equal to FD_SETSIZE.

[EINVAL] One of the specified file descriptors refers to a STREAM or multiplexer that is linked (directly or indirectly) downstream from a multiplexer.
APPLICATION USAGE

The use of a timeout does not affect any pending timers set up by `alarm()`, `ualarm()` or `settimer()`.

On successful completion, the object pointed to by the `timeout` argument may be modified.

SEE ALSO

`fcntl()`, `poll()`, `read()`, `write()`, `<sys/time.h>`.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME

semctl — semaphore control operations

SYNOPSIS

EX

```
#include <sys/sem.h>
int semctl(int semid, int semnum, int cmd, ...);
```

DESCRIPTION

The semctl() function provides a variety of semaphore control operations as specified by cmd. The fourth argument is optional and depends upon the operation requested. If required, it is of type union semun, which the application program must explicitly declare:

```
union semun {
    int val;
    struct semid_ds *buf;
    unsigned short *array;
} arg;
```

The following semaphore control operations as specified by cmd are executed with respect to the semaphore specified by semid and semnum. The level of permission required for each operation is shown with each command, see Section 2.6 on page 37. The symbolic names for the values of cmd are defined by the <sys/sem.h> header:

GETVAL

Return the value of semval, see <sys/sem.h>. Requires read permission.

SETVAL

Set the value of semval to arg.val, where arg is the value of the fourth argument to semctl(). When this command is successfully executed, the semadj value corresponding to the specified semaphore in all processes is cleared. Requires alter permission, see Section 2.6 on page 37.

GETPID

Return the value of sempid. Requires read permission.

GETNCNT

Return the value of semncnt. Requires read permission.

GETZCNT

Return the value of semzcnt. Requires read permission.

The following values of cmd operate on each semval in the set of semaphores:

GETALL

Return the value of semval for each semaphore in the semaphore set and place into the array pointed to by arg.array, where arg is the fourth argument to semctl(). Requires read permission.

SETALL

Set the value of semval for each semaphore in the semaphore set according to the array pointed to by arg.array, where arg is the fourth argument to semctl(). When this command is successfully executed, the semadj values corresponding to each specified semaphore in all processes are cleared. Requires alter permission.

The following values of cmd are also available:

IPC_STAT

Place the current value of each member of the semid_ds data structure associated with semid into the structure pointed to by arg.buf, where arg is the fourth argument to semctl(). The contents of this structure are defined in <sys/sem.h>. Requires read permission.
System Interfaces

**semctl()**

**IPC_SET**
Set the value of the following members of the semid_ds data structure associated with semid to the corresponding value found in the structure pointed to by arg.buf, where arg is the fourth argument to semctl():

- sem_perm.uid
- sem_perm.gid
- sem_perm.mode

The mode bits specified in Section 2.6.1 on page 37 are copied into the corresponding bits of the sem_perm.mode associated with semid. The stored values of any other bits are unspecified.

This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of sem_perm.cuid or sem_perm.uid in the semid_ds data structure associated with semid.

**IPC_RMID**
Remove the semaphore-identifier specified by semid from the system and destroy the set of semaphores and semid_ds data structure associated with it. This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of sem_perm.cuid or sem_perm.uid in the semid_ds data structure associated with semid.

**RETURN VALUE**
If successful, the value returned by semctl() depends on cmd as follows:

- GETVAL: The value of semval.
- GETPID: The value of sempid.
- GETNCNT: The value of semncnt.
- GETZCNT: The value of semzcnt.
- All others: 0.

Otherwise, semctl() returns −1 and errno indicates the error.

**ERRORS**
The semctl() function will fail if:

- [EACCES]: Operation permission is denied to the calling process, see Section 2.6 on page 37.
- [EINVAL]: The value of semid is not a valid semaphore identifier, or the value of semnum is less than 0 or greater than or equal to sem_nsems, or the value of cmd is not a valid command.
- [EPERM]: The argument cmd is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of sem_perm.cuid or sem_perm.uid in the data structure associated with semid.
- [ERANGE]: The argument cmd is equal to SETVAL or SETALL and the value to which semval is to be set is greater than the system-imposed maximum.

**APPLICATION USAGE**
The fourth parameter in the SYNOPSIS section is now specified as ... in order to avoid a clash with the ISO C standard when referring to the union semun (as defined in XPG3) and for
FUTURE DIRECTIONS
The IEEE 1003.4 standards committee is developing alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the routines described in this document can be easily modified to use alternative methods at a later date.

SEE ALSO
semget(), semop(), <sys/sem.h>, Section 2.6 on page 37.

CHANGE HISTORY
First released in Issue 2.
Derived from Issue 2 of the SVID.

Issue 4
The following changes are incorporated in this issue:

- The interface is no longer marked as OPTIONAL FUNCTIONALITY.
- Inclusion of the <sys/types.h> and <sys/ipc.h> headers is removed from the SYNOPSIS section.
- The last argument is now defined by an ellipsis symbol. In previous issues it was defined as a union of the various types required by settings of cmd. These are now defined individually in each description of permitted cmd settings. The text of the description of SETALL in the DESCRIPTION section now refers to the fourth argument instead of arg.buf.
- In the DESCRIPTION section the type of the array is specified in the descriptions of GETALL and SETALL.
- The [ENOSYS] error is removed from the ERRORS section.
- A FUTURE DIRECTIONS section is added warning application developers about migration to IEEE 1003.4 interfaces for interprocess communication.

Issue 4, Version 2
The fourth argument to semctl(), formerly specified in APPLICATION USAGE, is moved to the DESCRIPTION, and references to its elements are made more precise.
NAME
semget — get set of semaphores

SYNOPSIS
EX
#include <sys/sem.h>

int semget(key_t key, int nsems, int semflg);

DESCRIPTION
The semget() function returns the semaphore identifier associated with key.

A semaphore identifier with its associated semid_ds data structure and its associated set of nsems semaphores, see <sys/sem.h>, are created for key if one of the following is true:

- The argument key is equal to IPC_PRIVATE.
- The argument key does not already have a semaphore identifier associated with it and (semflg & IPC_CREAT) is non-zero.

Upon creation, the semid_ds data structure associated with the new semaphore identifier is initialised as follows:

- In the operation permissions structure sem_perm.cuid, sem_perm.uid, sem_perm.cgid and sem_perm.gid are set equal to the effective user ID and effective group ID, respectively, of the calling process.
- The low-order 9 bits of sem_perm.mode are set equal to the low-order 9 bits of semflg.
- The variable sem_nsems is set equal to the value of nsems.
- The variable sem_otime is set equal to 0 and sem_ctime is set equal to the current time.
- The data structure associated with each semaphore in the set is not initialised. The semctl() function with the command SETVAL or SETALL can be used to initialise each semaphore.

RETURN VALUE
Upon successful completion, semget() returns a non-negative integer, namely a semaphore identifier; otherwise, it returns -1 and errno will be set to indicate the error.

ERRORS
The semget() function will fail if:

[EACCES] A semaphore identifier exists for key, but operation permission as specified by the low-order 9 bits of semflg would not be granted. See Section 2.6 on page 37.

[EINVAL] The value of nsems is either less than or equal to 0 or greater than the system-imposed limit, or a semaphore identifier exists for the argument key, but the number of semaphores in the set associated with it is less than nsems and nsems is not equal to 0.

[ENOENT] A semaphore identifier does not exist for the argument key and (semflg & IPC_CREAT) is equal to 0.

[ENOSPC] A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphores system-wide would be exceeded.
FUTURE DIRECTIONS
The IEEE 1003.4 standards committee is developing alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the routines described in this document can be easily modified to use alternative methods at a later date.

SEE ALSO
semctl(), semop(), <sys/sem.h>, Section 2.6 on page 37.

CHANGE HISTORY
First released in Issue 2.
Derived from Issue 2 of the SVID.

Issue 4
The following changes are incorporated in this issue:

• The interface is no longer marked as OPTIONAL FUNCTIONALITY.

• Inclusion of the <sys/types.h> and <sys/ipc.h> headers is removed from the SYNOPSIS section.

• The [ENOSYS] error is removed from the ERRORS section.

• A FUTURE DIRECTIONS section is added warning application developers about migration to IEEE 1003.4 interfaces for interprocess communication.
NAME

semop — semaphore operations

SYNOPSIS

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

int semop(int semid, struct sembuf *sops, size_t nsops);
```

DESCRIPTION

The `semop()` function is used to perform atomically a user-defined array of semaphore operations on the set of semaphores associated with the semaphore identifier specified by the argument `semid`.

The argument `sops` is a pointer to a user-defined array of semaphore operation structures. The implementation will not modify elements of this array unless the application uses implementation-dependent extensions.

The argument `nsops` is the number of such structures in the array.

Each structure, `sembuf`, includes the following members:

<table>
<thead>
<tr>
<th>Member Type</th>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
<td>sem_num</td>
<td>semaphore number</td>
</tr>
<tr>
<td>short</td>
<td>sem_op</td>
<td>semaphore operation</td>
</tr>
<tr>
<td>short</td>
<td>sem_flg</td>
<td>operation flags</td>
</tr>
</tbody>
</table>

Each semaphore operation specified by `sem_op` is performed on the corresponding semaphore specified by `semid` and `sem_num`.

The variable `sem_op` specifies one of three semaphore operations:

1. If `sem_op` is a negative integer and the calling process has alter permission, one of the following will occur:
   - If `semval`, see `<sys/sem.h>`, is greater than or equal to the absolute value of `sem_op`, the absolute value of `sem_op` is subtracted from `semval`. Also, if `(sem_flg & SEM_UNDO)` is non-zero, the absolute value of `sem_op` is added to the calling process' `semadj` value for the specified semaphore.
   - If `semval` is less than the absolute value of `sem_op` and `(sem_flg & IPC_NOWAIT)` is non-zero, `semop()` will return immediately.
   - If `semval` is less than the absolute value of `sem_op` and `(sem_flg & IPC_NOWAIT)` is 0, `semop()` will increment the `semncnt` associated with the specified semaphore and suspend execution of the calling process until one of the following conditions occurs:
     - The value of `semval` becomes greater than or equal to the absolute value of `sem_op`. When this occurs, the value of `semcnt` associated with the specified semaphore is decremented, the absolute value of `sem_op` is subtracted from `semval` and, if `(sem_flg & SEM_UNDO)` is non-zero, the absolute value of `sem_op` is added to the calling process' `semadj` value for the specified semaphore.
     - The `semid` for which the calling process is awaiting action is removed from the system. When this occurs, `errno` is set equal to [EIDRM] and −1 is returned.
     - The calling process receives a signal that is to be caught. When this occurs, the value of `semcnt` associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in `sigaction()`.
2. If *sem_op* is a positive integer and the calling process has alter permission, the value of *sem_op* is added to *semval* and, if (*sem_flg* & SEM_UNDO) is non-zero, the value of *sem_op* is subtracted from the calling process’ *semadj* value for the specified semaphore.

3. If *sem_op* is 0 and the calling process has read permission, one of the following will occur:
   - If *semval* is 0, *semop()* will return immediately.
   - If *semval* is non-zero and (*sem_flg* & IPC_NOWAIT) is non-zero, *semop()* will return immediately.
   - If *semval* is non-zero and (*sem_flg* & IPC_NOWAIT) is 0, *semop()* will increment the *semzcnt* associated with the specified semaphore and suspend execution of the calling process until one of the following occurs:
     - The value of *semval* becomes 0, at which time the value of *semzcnt* associated with the specified semaphore is decremented.
     - The *semid* for which the calling process is awaiting action is removed from the system. When this occurs, *errno* is set equal to [EIDRM] and −1 is returned.
     - The calling process receives a signal that is to be caught. When this occurs, the value of *semzcnt* associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in *sigaction()*.

Upon successful completion, the value of *sempid* for each semaphore specified in the array pointed to by *sops* is set equal to the process ID of the calling process.

RETURN VALUE
Upon successful completion, *semop()* returns 0. Otherwise, it returns −1 and *errno* will be set to indicate the error.

ERRORS
The *semop()* function will fail if:

- **[E2BIG]** The value of *nsops* is greater than the system-imposed maximum.
- **[EACCES]** Operation permission is denied to the calling process, see Section 2.6 on page 37.
- **[EAGAIN]** The operation would result in suspension of the calling process but (*sem_flg* & IPC_NOWAIT) is non-zero.
- **[EFBIG]** The value of *sem_num* is less than 0 or greater than or equal to the number of semaphores in the set associated with *semid*.
- **[EIDRM]** The semaphore identifier *semid* is removed from the system.
- **[EINTR]** The *semop()* function was interrupted by a signal.
- **[EINVAL]** The value of *semid* is not a valid semaphore identifier, or the number of individual semaphores for which the calling process requests a SEM_UNDO would exceed the system-imposed limit.
- **[ENOSPC]** The limit on the number of individual processes requesting a SEM_UNDO would be exceeded.
- **[ERANGE]** An operation would cause a *semval* to overflow the system-imposed limit, or an operation would cause a *semadj* value to overflow the system-imposed limit.
FUTURE DIRECTIONS
The IEEE 1003.4 Standards Committee is developing alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the routines described in this document can be easily modified to use alternative methods at a later date.

SEE ALSO
exec, exit(), fork(), semctl(), semget(), <sys/ipc.h>, <sys/sem.h>, <sys/types.h>, Section 2.6 on page 37.

CHANGE HISTORY
First released in Issue 2.
Derived from Issue 2 of the SVID.

Issue 4
The following changes are incorporated in this issue:

• The interface is no longer marked as OPTIONAL FUNCTIONALITY.
• Inclusion of the <sys/types.h> and <sys/ipc.h> headers is removed from the SYNOPSIS section.
• The type of nsops is changed to size_t.
• The DESCRIPTION section is updated to indicate that an implementation will not modify the elements of sops unless the application uses implementation-dependent extensions.
• The [ENOSYS] error is removed from the ERRORS section.
• A FUTURE DIRECTIONS section is added warning application developers about migration to IEEE 1003.4 interfaces for interprocess communication.
NAME
setbuf — assign buffering to a stream

SYNOPSIS
#include <stdio.h>

void setbuf(FILE *stream, char *buf);

DESCRIPTION
Except that it returns no value, the function call:

    setbuf(stream, buf)

is equivalent to:

    setvbuf(stream, buf, _IOFBF, BUFSIZ)

if buf is not a null pointer, or to:

    setvbuf(stream, buf, _IONBF, BUFSIZ)

if buf is a null pointer.

RETURN VALUE
The setbuf() function returns no value.

ERRORS
No errors are defined.

APPLICATION USAGE
A common source of error is allocating buffer space as an “automatic” variable in a code block,
and then failing to close the stream in the same block.

With setbuf(), allocating a buffer of size bytes does not necessarily imply that all of size bytes are
used for the buffer area.

SEE ALSO
fopen(), setvbuf(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.
NAME
  setcontext — set current user context

SYNOPSIS
  #include <ucontext.h>

  int setcontext(const ucontext_t *ucp);

DESCRIPTION
  Refer to getcontext().

CHANGE HISTORY
  First released in Issue 4, Version 2.
NAME
setgid — set-group-ID

SYNOPSIS
#include <sys/types.h>
#include <unistd.h>
int setgid(gid_t gid);

DESCRIPTION
If the process has appropriate privileges, `setgid()` sets the real group ID, effective group ID and the saved set-group-ID to `gid`.

If the process does not have appropriate privileges, but `gid` is equal to the real group ID or the saved set-group-ID, `setgid()` function sets the effective group ID to `gid`; the real group ID and saved set-group-ID remain unchanged.

Any supplementary group IDs of the calling process remain unchanged.

RETURN VALUE
Upon successful completion, 0 is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.

ERRORS
The `setgid()` function will fail if:

[EINVAL] The value of the `gid` argument is invalid and is not supported by the implementation.

[EPERM] The process does not have appropriate privileges and `gid` does not match the real group ID or the saved set-group-ID.

SEE ALSO
`exec`, `getgid()`, `setuid()`, `<sys/types.h>`, `<unistd.h>`.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the FIPS requirements:

- All references to the saved set-user-ID are marked as extensions. This is because Issue 4 defines this mechanism as mandatory, whereas the ISO POSIX-1 standard defines that it is only supported if [POSIX_SAVED_IDS] is set.

Another change is incorporated as follows:

- The header `<sys/types.h>` is now marked as optional (OH); this header need not be included on XSI-conformant systems.
NAME
setgrent — reset group database to first entry

SYNOPSIS
#include <grp.h>

void setgrent(void);

DESCRIPTION
Refer to endgrent().

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
     setitimer — set value of interval timer

SYNOPSIS
     #include <sys/time.h>

     int setitimer(int which, const struct itimerval *value,
                   struct itimerval *ovalue);

DESCRIPTION
     Refer to getitimer().

CHANGE HISTORY
     First released in Issue 4, Version 2.
NAME

_setjmp — set jump point for a non-local goto

SYNOPSIS

UX

#include <setjmp.h>

int _setjmp(jmp_buf env);

DESCRIPTION

Refer to _longjmp().

CHANGE HISTORY

First released in Issue 4, Version 2.
setjmp( )

BASE

System Interfaces

NAME
setjmp — set jump point for a non-local goto
SYNOPSIS
#include <setjmp.h>
int setjmp(jmp_buf env);
DESCRIPTION
A call to setjmp( ), saves the calling environment in its env argument for later use by longjmp( ).
It is unspecified whether setjmp( ) is a macro or a function. If a macro definition is suppressed in
order to access an actual function, or a program defines an external identifier with the name
setjmp the behaviour is undefined.
All accessible objects have values as of the time longjmp( ) was called, except that the values of
objects of automatic storage duration which are local to the function containing the invocation of
the corresponding setjmp( ) which do not have volatile-qualified type and which are changed
between the setjmp( ) invocation and longjmp( ) call are indeterminate.
An invocation of setjmp( ) must appear in one of the following contexts only:
•

the entire controlling expression of a selection or iteration statement

•

one operand of a relational or equality operator with the other operand an integral constant
expression, with the resulting expression being the entire controlling expression of a
selection or iteration statement

•

the operand of a unary "!" operator with the resulting expression being the entire controlling
expression of a selection or iteration

•

the entire expression of an expression statement (possibly cast to void).

RETURN VALUE
If the return is from a direct invocation, setjmp( ) returns 0. If the return is from a call to
longjmp( ), setjmp( ) returns a non-zero value.
ERRORS
No errors are defined.
APPLICATION USAGE
In general, sigsetjmp( ) is more useful in dealing with errors and interrupts encountered in a lowlevel subroutine of a program.
SEE ALSO
longjmp( ), sigsetjmp( ), <setjmp.h>.
CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.
Issue 4
The following changes are incorporated in this issue:

520

•

This issue states that setjmp( ) is a macro or a function; previous issues stated that it was a
macro. Warnings have also been added about the suppression of a setjmp( ) macro definition.

•

Text describing the accessibility of objects after a longjmp( ) call is added to the
DESCRIPTION section. This text is imported from the entry for longjmp( ).

X/Open CAE Specification (1994)


• Text describing the contexts in which calls to \texttt{setjmp()} are valid is moved to the \texttt{DESCRIPTION} section from the \texttt{APPLICATION USAGE} section.

• The \texttt{APPLICATION USAGE} section is changed to refer to \texttt{sigsetjmp()}.
NAME
setkey — set encoding key (OPTIONAL FUNCTIONALITY)

SYNOPSIS

```c
#include <stdlib.h>

void setkey(const char * key);
```

DESCRIPTION

The `setkey()` function provides (rather primitive) access to an implementation-dependent encoding algorithm. The argument of `setkey()` is an array of length 64 bytes containing only the bytes with numerical value of 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored; this gives a 56-bit key which is used by the algorithm. This is the key that will be used with the algorithm to encode a string `block` passed to `encrypt()`.

RETURN VALUE

No values are returned.

ERRORS

The `setkey()` function will fail if:

- `[ENOSYS]` The functionality is not supported on this implementation.

APPLICATION USAGE

In some environments, decoding may not be implemented. This is related to U.S. Government restrictions on encryption and decryption routines: the DES decryption algorithm cannot be exported outside the U.S.A. Historical practice has been to ship a different version of the encryption library without the decryption feature in the routines supplied. Thus the exported version of `encrypt()` does encoding but not decoding.

Because `setkey()` does not return a value, applications wishing to check for errors should set `errno` to 0, call `setkey()`, then test `errno` and, if it is non-zero, assume an error has occurred.

SEE ALSO

`crypt()`, `encrypt()`, `<stdlib.h>`.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated in this issue:

- The type of argument `key` is changed from `char *` to `const char *`.
- The description of the array is put in terms of bytes instead of characters.
- The APPLICATION USAGE section is added.
NAME
setlocale — set program locale

SYNOPSIS
#include <locale.h>
char *setlocale(int category, const char *locale);

DESCRIPTION
The setlocale() function selects the appropriate piece of the program's locale, as specified by the category and locale arguments, and may be used to change or query the program's entire locale or portions thereof. The value LC_ALL for category names the program's entire locale; other values for category name only a part of the program's locale:

- **LC_COLLATE** Affects the behaviour of regular expressions and the collation functions.
- **LC_CTYPE** Affects the behaviour of regular expressions, character classification, character conversion functions, and wide character functions.
- **LC_MESSAGES** Affects what strings are expected by commands and utilities as affirmative or negative responses, what strings are given by commands and utilities as affirmative or negative responses, and the content of messages.
- **LC_MONETARY** Affects the behaviour of functions that handle monetary values.
- **LC_NUMERIC** Affects the radix character for the formatted input/output functions and the string conversion functions.
- **LC_TIME** Affects the behaviour of the time conversion functions.

The locale argument is a pointer to a character string containing the required setting of category. The contents of this string are implementation-dependent. In addition, the following preset values of locale are defined for all settings of category:

- **"POSIX"** Specifies the minimal environment for C-language translation called POSIX locale. If setlocale() is not invoked, the POSIX locale is the default.
- **"C"** Same as POSIX.
- **" "** Specifies an implementation-dependent native environment. For XSI-conformant systems, this corresponds to the value of the associated environment variables, LC_ * and LANG; see the XBD specification, Chapter 5, Locale and the XBD specification, Chapter 6, Environment Variables.

A null pointer

Used to direct setlocale() to query the current internationalised environment and return the name of the locale().

RETURN VALUE
Upon successful completion, setlocale() returns the string associated with the specified category for the new locale. Otherwise, setlocale() returns a null pointer and the program's locale is not changed.

A null pointer for locale causes setlocale() to return a pointer to the string associated with the category for the program's current locale. The program's locale is not changed.

The string returned by setlocale() is such that a subsequent call with that string and its associated category will restore that part of the program's locale. The string returned must not be modified by the program, but may be overwritten by a subsequent call to setlocale().
APPLICATION USAGE

The following code illustrates how a program can initialise the international environment for one language, while selectively modifying the program's locale such that regular expressions and string operations can be applied to text recorded in a different language:

```c
setlocale(LC_ALL, "De");
setlocale(LC_COLLATE, "Fr@dict");
```

Internationalised programs must call `setlocale()` to initiate a specific language operation. This can be done by calling `setlocale()` as follows:

```c
setlocale(LC_ALL, " ");
```

Changing the setting of LC_MESSAGES has no effect on catalogues that are already opened by calls to `catopen()`.

ERRORS

No errors are defined.

SEE ALSO

`exec`, `isalnum()`, `isalpha()`, `iscntrl()`, `isgraph()`, `islower()`, `isprint()`, `ispunct()`, `isspace()`, `isupper()`, `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`, `iswspace()`, `strcoll()`, `strerror()`, `strfmon()`, `strtod()`, `strxfrm()`, `tolower()`, `toupper()`, `towlower()`, `towupper()`, `wcscoll()`, `wcstod()`, `wcstombs()`, `wcsxfrm()`, `wctomb()`, `<langinfo.h>`, `<locale.h>`.

CHANGE HISTORY

First released in Issue 3.

Issue 4

The following changes are incorporated for alignment with the ISO C standard and the ISO POSIX-1 standard:

- The type of the argument `locale` is changed from `char *` to `const char *`.
- The name POSIX is added to the list of standard locale names.

The following change is incorporated for alignment with the ISO POSIX-2 standard:

- The LC_MESSAGES value for `category` is added to the DESCRIPTION section.

Other changes are incorporated as follows:

- The description of LC_MESSAGES is extended to indicate that this category also determines what strings are produced by commands and utilities for affirmative and negative responses, and that it affects the content of other program messages. This is marked as an extension.
- References to `nl_langinfo()` are removed.
- The description of the implementation-dependent native locale ("") is clarified by stating the related environment variables explicitly.
- The APPLICATION USAGE section is expanded.
NAME
setlogmask — set log priority mask

SYNOPSIS
UX
#include <syslog.h>

int setlogmask(int maskpri);

DESCRIPTION
Refer to closelog().

CHANGE HISTORY
First released in Issue 4, Version 2.
setpgid()  

NAME

setpgid — set process group ID for job control

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>

int setpgid(pid_t pid, pid_t pgid);
```

DESCRIPTION

The setpgid() function is used either to join an existing process group or create a new process group within the session of the calling process. The process group ID of a session leader will not change. Upon successful completion, the process group ID of the process with a process ID that matches pid will be set to pgid. As a special case, if pid is 0, the process ID of the calling process will be used. Also, if pgid is 0, the process group ID of the indicated process will be used.

RETURN VALUE

Upon successful completion, setpgid() returns 0. Otherwise −1 is returned and errno is set to indicate the error.

ERRORS

The setpgid() function will fail if:

- [EACCES] The value of the pid argument matches the process ID of a child process of the calling process and the child process has successfully executed one of the exec functions.

- [EINVAL] The value of the pgid argument is less than 0, or is not a value supported by the implementation.

- [EPERM] The process indicated by the pid argument is a session leader.

  The value of the pid argument matches the process ID of a child process of the calling process and the child process is not in the same session as the calling process.

  The value of the pgid argument is valid but does not match the process ID of the process indicated by the pid argument and there is no process with a process group ID that matches the value of the pgid argument in the same session as the calling process.

- [ESRCH] The value of the pid argument does not match the process ID of the calling process or of a child process of the calling process.

SEE ALSO

exec, getpgrp(), setsid(), tcsetpgrp(), <sys/types.h>, <unistd.h>.

CHANGE HISTORY

First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

Issue 4

The following changes are incorporated in this issue:

- The interface is no longer marked as OPTIONAL FUNCTIONALITY.

- The header <sys/types.h> is now marked as optional (oh); this header need not be included on XSI-conformant systems.
• The header `<unistd.h>` is added to the SYNOPSIS section.

• The DESCRIPTION in Issue 3 defined the behaviour of this function for implementations that either supported or did not support job control. As job control is defined as mandatory in Issue 4, only the former of these is now described.

• The [ENOSYS] error is removed from the ERRORS section.
NAME
setpgrp — set process group ID

SYNOPSIS
UX
#include <unistd.h>

pid_t setpgrp(void);

DESCRIPTION
If the calling process is not already a session leader, setpgrp() sets the process group ID of the
calling process to the process ID of the calling process. If setpgrp() creates a new session, then
the new session has no controlling terminal.

The setpgrp() function has no effect when the calling process is a session leader.

RETURN VALUE
Upon successful completion, setpgrp() returns the new process group ID.

ERRORS
No errors are defined.

SEE ALSO
exec, fork(), getpid(), getsid(), kill(), setsid(), <unistd.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
  setpriority — set process scheduling priority

SYNOPSIS
  #include <sys/resource.h>
  int setpriority(int which, id_t who, int priority);

DESCRIPTION
  Refer to getpriority().

CHANGE HISTORY
  First released in Issue 4, Version 2.
NAME
setregid — set real and effective group IDs

SYNOPSIS
#include <unistd.h>

int setregid(gid_t rgid, gid_t egid);

DESCRIPTION
The setregid() function is used to set the real and effective group IDs of the calling process. If
rgid is −1, the real group ID is not changed; if egid is −1, the effective group ID is not changed.
The real and effective group IDs may be set to different values in the same call.

Only a process with appropriate privileges can set the real group ID and the effective group ID
to any valid value.

A non-privileged process can set either the real group ID to the saved set-group-ID from execv(),
or the effective group ID to the saved set-group-ID or the real group ID.

Any supplementary group IDs of the calling process remain unchanged.

RETURN VALUE
Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to indicate
the error and neither of the group IDs will be changed.

ERRORS
The setregid() function will fail if:

[EINVAL] The value of the rgid or egid argument is invalid or out-of-range.
[EPERM] The process does not have appropriate privileges and a change other than
changing the real group ID to the saved set-group-ID, or changing the
effective group ID to the real group ID or the saved group ID, was requested.

APPLICATION USAGE
If a set-group-ID process sets its effective group ID to its real group ID, it can still set its effective
group ID back to the saved set-group-ID.

SEE ALSO
exec, getuid(), setreuid(), setuid(), <unistd.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
    setreuid — set real and effective user IDs

SYNOPSIS
    #include <unistd.h>

    int setreuid(uid_t ruid, uid_t euid);

DESCRIPTION
    The `setreuid()` function sets the real and effective user IDs of the current process to the values
    specified by the `ruid` and `euid` arguments. If `ruid` or `euid` is −1, the corresponding effective or real
    user ID of the current process is left unchanged.

    A process with appropriate privileges can set either ID to any value. An unprivileged process
    can only set the effective user ID if the `euid` argument is equal to either the real, effective, or
    saved user ID of the process.

    It is unspecified whether a process without appropriate privileges is permitted to change the real
    user ID to match the current real, effective or saved user ID of the process.

RETURN VALUE
    Upon successful completion, 0 is returned. Otherwise, −1 is returned and `errno` is set to indicate
    the error.

ERRORS
    The `setreuid()` function will fail if:

    [EINVAL] The value of the `ruid` or `euid` argument is invalid or out-of-range.

    [EPERM] The current process does not have appropriate privileges, and either an
            attempt was made to change the effective user ID to a value other than the
            real user ID or the saved set-user-ID or an an attempt was made to change the
            real user ID to a value not permitted by the implementation.

SEE ALSO
    `getuid()`, `setuid()`, `<unistd.h>`.

CHANGE HISTORY
    First released in Issue 4, Version 2.
NAME
setrlimit — control maximum resource consumption

SYNOPSIS
UX

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

int setrlimit(int resource, const struct rlimit *rlp);
```

DESCRIPTION
Refer to getrlimit().

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME

setsid — create session and set process group ID

SYNOPSIS

OH

```c
#include <sys/types.h>
#include <unistd.h>

pid_t setsid(void);
```

DESCRIPTION

The `setsid()` function creates a new session, if the calling process is not a process group leader. Upon return the calling process will be the session leader of this new session, will be the process group leader of a new process group, and will have no controlling terminal. The process group ID of the calling process will be set equal to the process ID of the calling process. The calling process will be the only process in the new process group and the only process in the new session.

RETURN VALUE

Upon successful completion, `setsid()` returns the value of the process group ID of the calling process. Otherwise it returns `(-1)` and sets `errno` to indicate the error.

ERRORS

The `setsid()` function will fail if:

- `[EPERM]` The calling process is already a process group leader, or the process group ID of a process other than the calling process matches the process ID of the calling process.

SEE ALSO

`getsid()`, `setpgid()`, `setpgrp()`, `<sys/types.h>`, `<unistd.h>`.

CHANGE HISTORY

First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

Issue 4

The following changes are incorporated in this issue:

- The header `<sys/types.h>` is now marked as optional (OH); this header need not be included on XSI-conformant systems.
- The header `<unistd.h>` is added to the `SYNOPSIS` section.
- The argument list is explicitly defined as `void`. 
NAME
setstate — switch pseudorandom number generator state arrays

SYNOPSIS
UX
#include <stdlib.h>

char *setstate(const char *state);

DESCRIPTION
Refer to initstate().

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
setuid — set-user-ID

SYNOPSIS
OH
#include <sys/types.h>
#include <unistd.h>

int setuid(uid_t uid);

DESCRIPTION
If the process has appropriate privileges, `setuid()` sets the real user ID, effective user ID, and the saved set-user-ID to `uid`.

If the process does not have appropriate privileges, but `uid` is equal to the real user ID or the saved set-user-ID, `setuid()` sets the effective user ID to `uid`; the real user ID and saved set-user-ID remain unchanged.

RETURN VALUE
Upon successful completion, 0 is returned. Otherwise, −1 is returned and `errno` is set to indicate the error.

ERRORS
The `setuid()` function will fail and return −1 and set `errno` to the corresponding value if one or more of the following are true:

[EINVAL] The value of the `uid` argument is invalid and not supported by the implementation.

[EPERM] The process does not have appropriate privileges and `uid` does not match the real user ID or the saved set-user-ID.

SEE ALSO
`exec`, `geteuid()`, `getuid()`, `setgid()`, `<sys/types.h>`, `<unistd.h>`.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the FIPS requirements:

- All references to the saved set-user-ID are marked as extensions. This is because Issue 4 defines this mechanism as mandatory, whereas the ISO POSIX-1 standard defines that it is only supported if [POSIX_SAVED_IDS] is set.

Other changes are incorporated as follows:

- The header `<sys/types.h>` is now marked as optional (OH); this header need not be included on XSI-conformant systems.
- The header `<unistd.h>` is added to the SYNOPSIS section.
NAME
setuxent — reset user accounting database to first entry

SYNOPSIS
UX
#include <utmpx.h>

void setuxent(void);

DESCRIPTION
Refer to endutxent().

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME

setvbuf — assign buffering to a stream

SYNOPSIS

#include <stdio.h>

int setvbuf(FILE *stream, char *buf, int type, size_t size);

DESCRIPTION

The setvbuf() function may be used after the stream pointed to by stream is associated with an open file but before any other operation is performed on the stream. The argument type determines how stream will be buffered, as follows: _IOFBF causes input/output to be fully buffered; _IOLBF causes input/output to be line buffered; _IONBF causes input/output to be unbuffered. If buf is not a null pointer, the array it points to may be used instead of a buffer allocated by setvbuf(). The argument size specifies the size of the array. The contents of the array at any time are indeterminate.

For information about streams, see Section 2.4 on page 32.

RETURN VALUE

Upon successful completion, setvbuf() returns 0. Otherwise, it returns a non-zero value if an invalid value is given for type or if the request cannot be honoured.

ERRORS

The setvbuf() function may fail if:

EX

[EBADF] The file descriptor underlying stream is not valid.

APPLICATION USAGE

A common source of error is allocating buffer space as an “automatic” variable in a code block, and then failing to close the stream in the same block.

With setvbuf(), allocating a buffer of size bytes does not necessarily imply that all of size bytes are used for the buffer area.

Applications should note that many implementations only provide line buffering on input from terminal devices.

SEE ALSO

fopen(), setbuf(), <stdio.h>.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following change is incorporated for alignment with the ISO C standard:

• This function is no longer marked as an extension.

Other changes are incorporated as follows:

• The second paragraph of the DESCRIPTION section is now in Section 2.4 on page 32.

• The [EBADF] error is marked as an extension.

• The APPLICATION USAGE section is expanded.
shmat()  SHARED MEMORY  System Interfaces

NAME
shmat — shared memory attach operation

SYNOPSIS
#include <sys/shm.h>

void *shmat(int shmid, const void * shmaddr, int shmflg);

DESCRIPTION
The shmat() function attaches the shared memory segment associated with the shared memory
identifier specified by shmid to the address space of the calling process. The segment is attached
at the address specified by one of the following criteria:
- If shmaddr is a null pointer, the segment is attached at the first available address as selected
  by the system.
- If shmaddr is not a null pointer and (shmflg & SHM_RND) is non-zero, the segment is attached
  at the address given by (shmaddr - ((ptrdiff_t)shmaddr % SHMLBA)) The character % is the C-
  language remainder operator.
- If shmaddr is not a null pointer and (shmflg & SHM_RND) is 0, the segment is attached at the
  address given by shmaddr.
- The segment is attached for reading if (shmflg & SHM_RDONLY) is non-zero and the calling
  process has read permission; otherwise, if it is 0 and the calling process has read and write
  permission, the segment is attached for reading and writing.

RETURN VALUE
Upon successful completion, shmat() increments the value of shm_nattach in the data structure
associated with the shared memory ID of the attached shared memory segment and returns the
segment’s start address.
Otherwise, the shared memory segment is not attached, shmat() returns −1 and errno is set to
indicate the error.

ERRORS
The shmat() function will fail if:
[EACCES] Operation permission is denied to the calling process, see Section 2.6 on page
37.
[EINVAL] The value of shmid is not a valid shared memory identifier; the shmaddr is not a
null pointer and the value of (shmaddr - ((ptrdiff_t)shmaddr % SHMLBA)) is an illegal address for attaching shared memory; or the shmaddr is not a null pointer, (shmflg & SHM_RND) is 0 and the value of shmaddr is an illegal address for attaching shared memory.
[EMFILE] The number of shared memory segments attached to the calling process
would exceed the system-imposed limit.
[ENOMEM] The available data space is not large enough to accommodate the shared
memory segment.
[ENOSYS] The function is not implemented.

FUTURE DIRECTIONS
The IEEE 1003.4 standards committee is developing alternative interfaces for interprocess
communication. Application developers who need to use IPC should design their applications
so that modules using the routines described in this document can be easily modified to use
alternative methods at a later date.
SEE ALSO

exec, exit(), fork(), shmctl(), shmdt(), shmget(), <sys/shm.h>, Section 2.6 on page 37.

CHANGE HISTORY

First released in Issue 2.
Derived from Issue 2 of the SVID.

Issue 4

The following changes are incorporated in this issue:

• The interface is no longer marked as OPTIONAL FUNCTIONALITY.
• Inclusion of the <sys/types.h> and <sys/ipc.h> headers is removed from the SYNOPSIS section.
• The type of argument shmaddr is changed from char * to const void*.
• The [ENOSYS] error is removed from the ERRORS section.
• The DESCRIPTION section is clarified in several places.
• A FUTURE DIRECTIONS section is added warning application developers about migration to IEEE 1003.4 interfaces for interprocess communication.
NAME
shmctl — shared memory control operations

SYNOPSIS

#include <sys/shm.h>

int shmctl(int shmid, int cmd, struct shmid_ds *buf);

DESCRIPTION
The shmctl() function provides a variety of shared memory control operations as specified by cmd. The following values for cmd are available:

IPC_STAT Place the current value of each member of the shmid_ds data structure associated with shmid into the structure pointed to by buf. The contents of the structure are defined in <sys/shm.h>.

IPC_SET Set the value of the following members of the shmid_ds data structure associated with shmid to the corresponding value found in the structure pointed to by buf:

- shm_perm.uid
- shm_perm.gid
- shm_perm.mode low-order nine bits

IPC_SET can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of shm_perm.cuid or shm_perm.uid in the shmid_ds data structure associated with shmid.

IPC_RMID Remove the shared memory identifier specified by shmid from the system and destroy the shared memory segment and shmid_ds data structure associated with it. IPC_RMID can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of shm_perm.cuid or shm_perm.uid in the shmid_ds data structure associated with shmid.

RETURN VALUE
Upon successful completion, shmctl() returns 0. Otherwise, it returns −1 and errno will be set to indicate the error.

ERRORS
The shmct1() function will fail if:

[EACCES] The argument cmd is equal to IPC_STAT and the calling process does not have read permission, see Section 2.6 on page 37.

EINVAL The value of shmid is not a valid shared memory identifier, or the value of cmd is not a valid command.

[ENOSYS] The function is not implemented.

[EPERM] The argument cmd is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of shm_perm.cuid or shm_perm.uid in the data structure associated with shmid.

The shmctl() function may fail if:

UX [EOVERFLOW] The cmd argument is IPC_STAT and the gid or uid value is too large to be stored in the structure pointed to by the buf argument.
FUTURE DIRECTIONS
The IEEE 1003.4 standards committee is developing alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the routines described in this document can be easily modified to use alternative methods at a later date.

SEE ALSO
shmctl(), shmdt(), shmget(), <sys/shm.h>, Section 2.6 on page 37.

CHANGE HISTORY
First released in Issue 2.
Derived from Issue 2 of the SVID.

Issue 4
The following changes are incorporated in this issue:
- The interface is no longer marked as OPTIONAL FUNCTIONALITY.
- Inclusion of the <sys/types.h> and <sys/ipc.h> headers is removed from the SYNOPSIS section.
- The [ENOSYS] error is removed from the ERRORS section.
- A FUTURE DIRECTIONS section is added warning application developers about migration to IEEE 1003.4 interfaces for interprocess communication.

Issue 4, Version 2
The ERRORS section is updated for X/OPEN UNIX conformance to include [EOVERFLOW] as an optional error.
NAME

shmdt — shared memory detach operation

SYNOPSIS

```c
#include <sys/shm.h>

int shmdt(const void *shmaddr);
```

DESCRIPTION

The `shmdt()` function detaches from the calling process’ address space the shared memory segment located at the address specified by `shmaddr`.

RETURN VALUE

Upon successful completion, `shmdt()` will decrement the value of `shm_nattach` in the data structure associated with the shared memory ID of the attached shared memory segment and return 0.

Otherwise, the shared memory segment will not be detached, `shmdt()` will return −1 and `errno` will be set to indicate the error.

ERRORS

The `shmdt()` function will fail if:

- [EINVAL] The value of `shmaddr` is not the data segment start address of a shared memory segment.
- [ENOSYS] The function is not implemented.

FUTURE DIRECTIONS

The IEEE 1003.4 Standards Committee is developing alternative interfaces for interprocess communication. Application developers who need to use IPC should design their Applications so that modules using the routines described in this document can be easily modified to use alternative methods at a later date.

SEE ALSO

`exec`, `exit()`, `fork()`, `shmat()`, `shmctl()`, `shmget()`, `<sys/shm.h>`, Section 2.6 on page 37.

CHANGE HISTORY

First released in Issue 2.

Derived from Issue 2 of the SVID.

Issue 4

The following changes are incorporated in this issue:

- The interface is no longer marked as OPTIONAL FUNCTIONALITY.
- Inclusion of the `<sys/types.h>` and `<sys/ipc.h>` headers is removed from the SYNOPSIS section.
- The type of argument `shmaddr` is changed from `char *` to `const void *`.
- The DESCRIPTION section is clarified in several places.
- The [ENOSYS] error is removed from the ERRORS section.
- A FUTURE DIRECTIONS section is added warning application developers about migration to IEEE 1003.4 interfaces for interprocess communication.
NAME

shmget — get shared memory segment

SYNOPSIS

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

int shmget(key_t key, size_t size, int shmflg);
```

DESCRIPTION

The `shmget()` function returns the shared memory identifier associated with `key`.

A shared memory identifier, associated data structure and shared memory segment of at least `size` bytes, see `<sys/shm.h>`, are created for `key` if one of the following is true:

- The argument `key` is equal to IPC_PRIVATE.
- The argument `key` does not already have a shared memory identifier associated with it and `(shmflg & IPC_CREAT)` is non-zero.

Upon creation, the data structure associated with the new shared memory identifier is initialised as follows:

- The value of `shm_perm.cuid`, `shm_perm.uid`, `shm_perm.cgid` and `shm_perm.gid` are set equal to the effective user ID and effective group ID, respectively, of the calling process.
- The low-order nine bits of `shm_perm.mode` are set equal to the low-order nine bits of `shmflg`. The value of `shm_segsz` is set equal to the value of `size`.
- The values of `shm_lpid`, `shm_nattch`, `shm_atime` and `shm_dtime` are set equal to 0.
- The value of `shm_ctime` is set equal to the current time.

RETURN VALUE

Upon successful completion, `shmget()` returns a non-negative integer, namely a shared memory identifier; otherwise, it returns −1 and `errno` will be set to indicate the error.

ERRORS

The `shmget()` function will fail if:

- **[EACCES]** A shared memory identifier exists for `key` but operation permission as specified by the low-order nine bits of `shmflg` would not be granted. See Section 2.6 on page 37.
- **[EEXIST]** A shared memory identifier exists for the argument `key` but `(shmflg & IPC_CREAT) && (shmflg & IPC_EXCL)` is non-zero.
- **[EINVAL]** The value of `size` is less than the system-imposed minimum or greater than the system-imposed maximum, or a shared memory identifier exists for the argument `key` but the size of the segment associated with it is less than `size` and `size` is not 0.
- **[ENOENT]** A shared memory identifier does not exist for the argument `key` and `(shmflg & IPC_CREAT)` is 0.
- **[ENOMEM]** A shared memory identifier and associated shared memory segment are to be created but the amount of available physical memory is not sufficient to fill the request.
- **[ENOSPC]** A shared memory identifier is to be created but the system-imposed limit on the maximum number of allowed shared memory identifiers system-wide would be exceeded.
The function is not implemented.

**FUTURE DIRECTIONS**

The IEEE 1003.4 standards committee is developing alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the routines described in this document can be easily modified to use alternative methods at a later date.

**SEE ALSO**

`shmat()`, `shmctl()`, `shmdt()`, `<sys/shm.h>`, Section 2.6 on page 37.

**CHANGE HISTORY**

First released in Issue 2.

Derived from Issue 2 of the SVID.

**Issue 4**

The following changes are incorporated in this issue:

- The interface is no longer marked as OPTIONAL FUNCTIONALITY.
- Inclusion of the `<sys/types.h>` and `<sys/ipc.h>` headers is removed from the SYNOPSIS section.
- The [ENOSYS] error is removed from the ERRORS section.
- A FUTURE DIRECTIONS section is added warning application developers about migration to IEEE 1003.4 interfaces for interprocess communication.
NAME

sigaction — examine and change signal action

SYNOPSIS

```c
#include <signal.h>

int sigaction(int sig, const struct sigaction *act,
               struct sigaction *oact);
```

DESCRIPTION

The `sigaction()` function allows the calling process to examine and/or specify the action to be associated with a specific signal. The argument `sig` specifies the signal; acceptable values are defined in `<signal.h>`.

The structure `sigaction`, used to describe an action to be taken, is defined in the header `<signal.h>` to include at least the following members:

<table>
<thead>
<tr>
<th>Member Type</th>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void(*) (int)</td>
<td>sa_handler</td>
<td>SIG_DFL, SIG_IGN or pointer to a function.</td>
</tr>
<tr>
<td>sigset_t</td>
<td>sa_mask</td>
<td>Additional set of signals to be blocked during execution of signal-catching function.</td>
</tr>
<tr>
<td>int</td>
<td>sa_flags</td>
<td>Special flags to affect behaviour of signal.</td>
</tr>
<tr>
<td>UX</td>
<td>void(*)(int, siginfo_t *, void *)</td>
<td>Signal-catching function.</td>
</tr>
</tbody>
</table>

If the argument `act` is not a null pointer, it points to a structure specifying the action to be associated with the specified signal. If the argument `oact` is not a null pointer, the action previously associated with the signal is stored in the location pointed to by the argument `oact`. If the argument `act` is a null pointer, signal handling is unchanged; thus, the call can be used to enquire about the current handling of a given signal. The `sa_handler` field of the `sigaction` structure identifies the action to be associated with the specified signal. If the `sa_handler` field specifies a signal-catching function, the `sa_mask` field identifies a set of signals that will be added to the process’ signal mask before the signal-catching function is invoked. The SIGKILL and SIGSTOP signals will not be added to the signal mask using this mechanism; this restriction will be enforced by the system without causing an error to be indicated.

The `sa_flags` field can be used to modify the behaviour of the specified signal.

The following flags, defined in the header `<signal.h>`, can be set in `sa_flags`:

- **SA_NOCLDSTOP**: Do not generate SIGCHLD when children stop.
- **SA_ONSTACK**: If set and an alternate signal stack has been declared with `sigaltstack()` or `sigstack()`, the signal will be delivered to the calling process on that stack. Otherwise, the signal will be delivered on the current stack.
- **SA_RESETHAND**: If set, the disposition of the signal will be reset to SIG_DFL and the SA_SIGINFO flag will be cleared on entry to the signal handler (Note: SIGILL and SIGTRAP cannot be automatically reset when delivered; the system silently enforces this restriction). Otherwise, the disposition of the signal will not be modified on entry to the signal handler.

In addition, if this flag is set, `sigaction()` behaves as if the SA_NODEFER flag were also set.

- **SA_RESTART**: This flag affects the behaviour of interruptible functions; that is, those specified to fail with `errno` set to [EINTR]. If set, and a function specified...
as interruptible is interrupted by this signal, the function will restart and
will not fail with [EINTR] unless otherwise specified. If the flag is not set,
interruptible functions interrupted by this signal will fail with errno set to
[EINTR].

SA_SIGINFO
If cleared and the signal is caught, the signal-catching function will be entered as:

    void func(int signo);

where signo is the only argument to the signal catching function. In this case the sa_handler member must be used to describe the signal catching function and the application must not modify the sa_sigaction member.

If SA_SIGINFO is set and the signal is caught, the signal-catching function will be entered as:

    void func(int signo, siginfo_t *info, void *context);

where two additional arguments are passed to the signal catching function. If the second argument is not a null pointer, it will point to an object of type siginfo_t explaining the reason why the signal was generated; the third argument can be cast to a pointer to an object of type ucontext_t to refer to the receiving process’ context that was interrupted when the signal was delivered. In this case the sa_sigaction member must be used to describe the signal catching function and the application must not modify the sa_handler member.

The si_signo member contains the system-generated signal number.

The si_errno member may contain implementation-dependent additional error information; if non-zero, it contains an error number identifying the condition that caused the signal to be generated.

The si_code member contains a code identifying the cause of the signal. If the value of si_code is less than or equal to 0, then the signal was generated by a process and si_pid and si_uid respectively indicate the process ID and the real user ID of the sender. The values of si_pid and si_uid are otherwise meaningless.

SA_NOCLDWAIT
If set, and sig equals SIGCHLD, child processes of the calling processes will not be transformed into zombie processes when they terminate. If the calling process subsequently waits for its children, and the process has no unwaited for children that were transformed into zombie processes, it will block until all of its children terminate, and wait(), wait3(), waitid() and waitpid() will fail and set errno to [ECHILD]. Otherwise, terminating child processes will be transformed into zombie processes, unless SIGCHLD is set to SIG_IGN.

SA_NODEFER
If set and sig is caught, sig will not be added to the process’ signal mask on entry to the signal handler unless it is included in sa_mask. Otherwise, sig will always be added to the process’ signal mask on entry to the signal handler.
If **sig** is SIGCHLD and the SA_NOCLDSTOP flag is not set in **sa_flags**, and the implementation supports the SIGCHLD signal, then a SIGCHLD signal will be generated for the calling process whenever any of its child processes stop. If **sig** is SIGCHLD and the SA_NOCLDSTOP flag is set in **sa_flags**, then the implementation will not generate a SIGCHLD signal in this way.

When a signal is caught by a signal-catching function installed by **sigaction()** , a new signal mask is calculated and installed for the duration of the signal-catching function (or until a call to either **sigprocmask()** or **sigsuspend()** is made). This mask is formed by taking the union of the current signal mask and the value of the **sa_mask** for the signal being delivered unless SA_NODEFER or SA_RESETHAND is set, and then including the signal being delivered. If and when the user’s signal handler returns normally, the original signal mask is restored.

Once an action is installed for a specific signal, it remains installed until another action is explicitly requested (by another call to **sigaction()**), until the SA_RESETHAND flag causes resetting of the handler, or until one of the exec functions is called.

If the previous action for **sig** had been established by **signal()** , the values of the fields returned in the structure pointed to by **oact** are unspecified, and in particular **oact->sa_handler** is not necessarily the same value passed to **signal()** . However, if a pointer to the same structure or a copy thereof is passed to a subsequent call to **sigaction()** via the **act** argument, handling of the signal will be as if the original call to **signal()** were repeated.

If **sigaction()** fails, no new signal handler is installed.

It is unspecified whether an attempt to set the action for a signal that cannot be caught or ignored to SIG_DFL is ignored or causes an error to be returned with **errno** set to [EINVAL].

A signal is said to be **generated** for (or sent to) a process when the event that causes the signal first occurs. Examples of such events include detection of hardware faults, timer expiration and terminal activity, as well as the invocation of **kill()** . In some circumstances, the same event generates signals for multiple processes.

Each process has an action to be taken in response to each signal defined by the system (see **Signal Actions** on page 548). A signal is said to be **delivered** to a process when the appropriate action for the process and signal is taken.

During the time between the generation of a signal and its delivery, the signal is said to be **pending** . Ordinarily, this interval cannot be detected by an application. However, a signal can be **blocked** from delivery to a process. If the action associated with a blocked signal is anything other than to ignore the signal, and if that signal is generated for the process, the signal will remain pending until either it is unblocked or the action associated with it is set to ignore the signal. If the action associated with a blocked signal is to ignore the signal and if that signal is generated for the process, it is unspecified whether the signal is discarded immediately upon generation or remains pending.

Each process has a **signal mask** that defines the set of signals currently blocked from delivery to it. The signal mask for a process is initialised from that of its parent. The **sigaction()** , **sigprocmask()** and **sigsuspend()** functions control the manipulation of the signal mask.

The determination of which action is taken in response to a signal is made at the time the signal is delivered, allowing for any changes since the time of generation. This determination is independent of the means by which the signal was originally generated. If a subsequent occurrence of a pending signal is generated, it is implementation-dependent as to whether the signal is delivered more than once. The order in which multiple, simultaneously pending signals are delivered to a process is unspecified.

When any stop signal (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) is generated for a process, any pending SIGCONT signals for that process will be discarded. Conversely, when SIGCONT is
generated for a process, all pending stop signals for that process will be discarded. When
SIGCONT is generated for a process that is stopped, the process will be continued, even if the
SIGCONT signal is blocked or ignored. If SIGCONT is blocked and not ignored, it will remain
pending until it is either unblocked or a stop signal is generated for the process.

An implementation will document any condition not specified by this document under which
the implementation generates signals.

Signal Actions

There are three types of action that can be associated with a signal: SIG_DFL, SIG_IGN or a
pointer to a function. Initially, all signals will be set to SIG_DFL or SIG_IGN prior to entry of the
main() routine (see the exec functions). The actions prescribed by these values are as follows:

SIG_DFL — signal-specific default action

- The default actions for the signals defined in this document are specified under
  <signal.h>.

- If the default action is to stop the process, the execution of that process is temporarily
  suspended. When a process stops, a SIGCHLD signal will be generated for its parent
  process, unless the parent process has set the SA_NOCLDSTOP flag. While a process is
  stopped, any additional signals that are sent to the process will not be delivered until the
  process is continued, except SIGKILL which always terminates the receiving process. A
  process that is a member of an orphaned process group will not be allowed to stop in
  response to the SIGTSTP, SIGTTIN or SIGTTOU signals. In cases where delivery of one
  of these signals would stop such a process, the signal will be discarded.

- Setting a signal action to SIG_DFL for a signal that is pending, and whose default action
  is to ignore the signal (for example, SIGCHLD), will cause the pending signal to be
  discarded, whether or not it is blocked.

SIG_IGN — ignore signal

- Delivery of the signal will have no effect on the process. The behaviour of a process is
  undefined after it ignores a SIGFPE, SIGILL or SIGSEGV signal that was not generated
  by kill() or raise().

- The system will not allow the action for the signals SIGKILL or SIGSTOP to be set to
  SIG_IGN.

- Setting a signal action to SIG_IGN for a signal that is pending will cause the pending
  signal to be discarded, whether or not it is blocked.

- If a process sets the action for the SIGCHLD signal to SIG_IGN, the behaviour is
  unspecified, except as specified below.

UX

If the action for the SIGCHLD signal is set to SIG_IGN, child processes of the calling
processes will not be transformed into zombie processes when they terminate. If the
calling process subsequently waits for its children, and the process has no unwaited for
children that were transformed into zombie processes, it will block until all of its
children terminate, and wait(), wait3(), waitid() and waitpid() will fail and set errno to
[ECHILD].

pointer to a function — catch signal

- On delivery of the signal, the receiving process is to execute the signal-catching function
  at the specified address. After returning from the signal-catching function, the receiving
  process will resume execution at the point at which it was interrupted.
### System Interfaces

**BASE**

#### sigaction()

**UX**

- If `SA_SIGINFO` is cleared, the signal-catching function will be entered as:

  ```c
  void func(int signo);
  ```

  where `func` is the specified signal-catching function and `signo` is the signal number of the signal being delivered.

- If `SA_SIGINFO` is set, the signal-catching function will be entered as:

  ```c
  void func(int signo, siginfo_t *siginfo, void *ucontextptr);
  ```

  where `func` is the specified signal-catching function, `signo` is the signal number of the signal being delivered, `siginfo` points to an object of type `siginfo_t` associated with the signal being delivered, and `ucontextptr` points to a `ucontext_t`.

**UX**

- The behaviour of a process is undefined after it returns normally from a signal-catching function for a `SIGBUS`, `SIGFPE`, `SIGILL` or `SIGSEGV` signal that was not generated by `kill()` or `raise()`.

- The system will not allow a process to catch the signals `SIGKILL` and `SIGSTOP`.

- If a process establishes a signal-catch function for the `SIGCHLD` signal while it has a terminated child process for which it has not waited, it is unspecified whether a `SIGCHLD` signal is generated to indicate that child process.

- When signal-catch functions are invoked asynchronously with process execution, the behaviour of some of the functions defined by this document is unspecified if they are called from a signal-catch function.

The following table defines a set of functions that are either reentrant or not interruptible by signals. Therefore applications may invoke them, without restriction, from signal-catch functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>access()</td>
<td>fstat()</td>
<td>read()</td>
<td>sysconf()</td>
</tr>
<tr>
<td>alarm()</td>
<td>getegid()</td>
<td>rename()</td>
<td>tcdrain()</td>
</tr>
<tr>
<td>cfgetispeed()</td>
<td>geteuid()</td>
<td>rmdir()</td>
<td>tcflow()</td>
</tr>
<tr>
<td>cfgetospeed()</td>
<td>getgid()</td>
<td>setgid()</td>
<td>tcflush()</td>
</tr>
<tr>
<td>cfsetispeed()</td>
<td>getgroups()</td>
<td>setpgid()</td>
<td>tcgetattr()</td>
</tr>
<tr>
<td>cfsetospeed()</td>
<td>getpgrp()</td>
<td>setsid()</td>
<td>tcgetpgrp()</td>
</tr>
<tr>
<td>chdir()</td>
<td>getpid()</td>
<td>setuid()</td>
<td>tcsetbreak()</td>
</tr>
<tr>
<td>chmod()</td>
<td>getppid()</td>
<td>sigaction()</td>
<td>tcsetattr()</td>
</tr>
<tr>
<td>chown()</td>
<td>getuid()</td>
<td>sigaddset()</td>
<td>tcsetpgrp()</td>
</tr>
<tr>
<td>close()</td>
<td>kill()</td>
<td>sigdelset()</td>
<td>time()</td>
</tr>
<tr>
<td>creat()</td>
<td>link()</td>
<td>sigemptyset()</td>
<td>times()</td>
</tr>
<tr>
<td>dup2()</td>
<td>lseek()</td>
<td>sigfillset()</td>
<td>umask()</td>
</tr>
<tr>
<td>dup()</td>
<td>mkdir()</td>
<td>sigismember()</td>
<td>unlink()</td>
</tr>
<tr>
<td>execl()</td>
<td>mknod()</td>
<td>signal()</td>
<td>unistd()</td>
</tr>
<tr>
<td>execve()</td>
<td>open()</td>
<td>sigpending()</td>
<td>utime()</td>
</tr>
<tr>
<td>_exit()</td>
<td>pathconf()</td>
<td>sigprocmask()</td>
<td>wait()</td>
</tr>
<tr>
<td>fcntl()</td>
<td>pause()</td>
<td>sigsuspend()</td>
<td>waitpid()</td>
</tr>
<tr>
<td>fork()</td>
<td>pipe()</td>
<td>sleep()</td>
<td>write()</td>
</tr>
<tr>
<td>fpathconf()</td>
<td>raise()</td>
<td>stat()</td>
<td></td>
</tr>
</tbody>
</table>

**EX**

All functions not in the above table are considered to be unsafe with respect to signals. In the presence of signals, all functions defined by this document will behave as defined when called from or interrupted by a signal-catch function, with a single exception: when a signal interrupts an unsafe function and the signal-catch function calls an
unsafe function, the behaviour is undefined.

Signal Effects on Other Functions
Signals affect the behaviour of certain functions defined by this document if delivered to a process while it is executing such a function. If the action of the signal is to terminate the process, the process will be terminated and the function will not return. If the action of the signal is to stop the process, the process will stop until continued or terminated. Generation of a SIGCONT signal for the process causes the process to be continued, and the original function will continue at the point the process was stopped. If the action of the signal is to invoke a signal-catching function, the signal-catching function will be invoked; in this case the original function is said to be interrupted by the signal. If the signal-catching function executes a return statement, the behaviour of the interrupted function will be as described individually for that function. Signals that are ignored will not affect the behaviour of any function; signals that are blocked will not affect the behaviour of any function until they are unblocked and then delivered.

RETURN VALUE
Upon successful completion, `sigaction()` returns 0. Otherwise −1 is returned, `errno` is set to indicate the error and no new signal-catching function will be installed.

ERRORS
The `sigaction()` function will fail if:

[EINVAL] The `sig` argument is not a valid signal number or an attempt is made to catch a signal that cannot be caught or ignore a signal that cannot be ignored.

The `sigaction()` function may fail if:

[EINVAL] An attempt was made to set the action to SIG_DFL for a signal that cannot be caught or ignored (or both).

APPLICATION USAGE
The `sigaction()` function supersedes the `signal()` interface, and should be used in preference. In particular, `sigaction()` and `signal()` should not be used in the same process to control the same signal. The behaviour of reentrant functions, as defined in the description, is as specified by this document, regardless of invocation from a signal-catching function. This is the only intended meaning of the statement that reentrant functions may be used in signal-catching functions without restrictions. Applications must still consider all effects of such functions on such things as data structures, files and process state. In particular, application writers need to consider the restrictions on interactions when interrupting `sleep()` and interactions among multiple handles for a file description. The fact that any specific function is listed as reentrant does not necessarily mean that invocation of that function from a signal-catching function is recommended.

In order to prevent errors arising from interrupting non-reentrant function calls, applications should protect calls to these functions either by blocking the appropriate signals or through the use of some programmatic semaphore. This document does not address the more general problem of synchronising access to shared data structures. Note in particular that even the “safe” functions may modify the global variable `errno`; the signal-catching function may want to save and restore its value. Naturally, the same principles apply to the reentrancy of application routines and asynchronous data access. Note that `longjmp()` and `siglongjmp()` are not in the list of reentrant functions. This is because the code executing after `longjmp()` and `siglongjmp()` can call any unsafe functions with the same danger as calling those unsafe functions directly from the signal handler. Applications that use `longjmp()` and `siglongjmp()` from within signal handlers require rigorous protection in order to be portable. Many of the other functions that are excluded from the list are traditionally implemented using either `malloc()` or `free()` functions or
the standard I/O library, both of which traditionally use data structures in a non-reentrant manner. Because any combination of different functions using a common data structure can cause reentrancy problems, this document does not define the behaviour when any unsafe function is called in a signal handler that interrupts an unsafe function.

If the signal occurs other than as the result of calling `abort()`, `kill()` or `raise()`, the behaviour is undefined if the signal handler calls any function in the standard library other than one of the functions listed in the table above or refers to any object with static storage duration other than by assigning a value to a static storage duration variable of type `volatile sig_atomic_t`. Furthermore, if such a call fails, the value of `errno` is indeterminate.

```
UX
```
Usually, the signal is executed on the stack that was in effect before the signal was delivered. An alternate stack may be specified to receive a subset of the signals being caught.

When the signal handler returns, the receiving process will resume execution at the point it was interrupted unless the signal handler makes other arrangements. If `longjmp()` or `_longjmp()` is used to leave the signal handler, then the signal mask must be explicitly restored by the process.

POSIX.4-1993 defines the third argument of a signal handling function when `SA_SIGINFO` is set as a `void *` instead of a `ucontext_t *`, but without requiring type checking. New applications should explicitly cast the third argument of the signal handling function to `ucontext_t *`.

The BSD optional four argument signal handling function is not supported by this specification. The BSD declaration would be

```
void handler(int sig, int code, struct sigcontext *scp, char *addr);
```

where `sig` is the signal number, `code` is additional information on certain signals, `scp` is a pointer to the sigcontext structure, and `addr` is additional address information. Much the same information is available in the objects pointed to by the second argument of the signal handler specified when `SA_SIGINFO` is set.

**FUTURE DIRECTIONS**
The `fpathconf()` function is marked as an extension in the list of safe functions because it is not included in the corresponding list in the ISO POSIX-1 standard, but it is expected to be added in a future revision of that standard.

**SEE ALSO**
`bsd_signal()`, `kill()`, `_longjmp()`, `longjmp()`, `raise()`, `sigaddset()`, `sigaltstack()`, `sigdelset()`, `sigemptyset()`, `sigfilset()`, `sigismember()`, `signal()`, `sigprocmask()`, `sigsuspend()`, `wait()`, `wait3()`, `waitid()`, `waitpid()`, `<signal.h>`, `<ucontext.h>`.

**CHANGE HISTORY**
First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

**Issue 4**
The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- The type of argument `act` is changed from `struct sigaction *` to `const struct sigaction *`.
- A statement is added to the `DESCRIPTION` section indicating that the consequence of attempting to set `SIG_DFL` for a signal that cannot be caught or ignored is unspecified. The `EINVAL` error, describing one possible reaction to this condition, is added to the `ERRORS` section.
Other changes are incorporated as follows:

- The `raise()` and `signal()` functions are added to the list of functions that are either reentrant or not interruptible by signals; `fpathconf()` is also added to this list and marked as an extension; `ustat()` is removed from the list, as this function is withdrawn from the interface definition. It is no longer specified whether `abort()`, `chroot()`, `exit()` and `longjmp()` also fall into this category of functions.

- The **APPLICATION USAGE** section is added. Most of this text is moved from the **DESCRIPTION SECTION** in Issue 3.

- The **FUTURE DIRECTIONS** section is added.

**Issue 4, Version 2**

The following changes are incorporated for X/OPEN UNIX conformance:

- The **DESCRIPTION** describes `sa_sigaction`, the member of the `sigaction` structure that is the signal-capturing function.

- The **DESCRIPTION** describes the `SA_ONSTACK`, `SA_RESETHAND`, `SA_RESTART`, `SA_SIGINFO`, `SA_NOCLDWAIT` and `SA_NODEFER` settings of `sa_flags`. The text describes the implications of the use of `SA_SIGINFO` for the number of arguments passed to the signal-capturing function. The text also describes the effects of the `SA_NODEFER` and `SA_RESETHAND` flags on the delivery of a signal and on the permanence of an installed action.

- The **DESCRIPTION** specifies the effect if the action for the SIGCHLD signal is set to `SIG_IGN`.

- In the **DESCRIPTION**, additional text describes the effect if the action is a pointer to a function. A new bullet covers the case where `SA_SIGINFO` is set. `SIGBUS` is given as an additional signal for which the behaviour of a process is undefined following a normal return from the signal-capturing function.

- The **APPLICATION USAGE** section is updated to describe use of an alternate signal stack; resumption of the process receiving the signal; coding for compatibility with POSIX.4-1993; and implementation of signal-handling functions in BSD.
NAME
sigaddset — add a signal to a signal set

SYNOPSIS
#include <signal.h>

int sigaddset(sigset_t * set, int signo);

DESCRIPTION
The sigaddset() function adds the individual signal specified by the signo to the signal set pointed to by set.

RETURN VALUE
Upon successful completion, sigaddset() returns 0. Otherwise, it returns −1 and sets errno to indicate the error.

ERRORS
The sigaddset() function may fail if:

EINVAL The value of the signo argument is an invalid or unsupported signal number.

APPLICATION USAGE
Applications should call either sigemptyset() or sigfillset() at least once for each object of type sigset_t prior to any other use of that object. If such an object is not initialised in this way, but is nonetheless supplied as an argument to any of sigaction(), sigaddset(), sigdelset(), sigismember(), sigpending() or sigprocmask(), the results are undefined.

SEE ALSO
sigaction(), sigdelset(), sigemptyset(), sigfillset(), sigismember(), sigpending(), sigprocmask(), sigsuspend(), <signal.h>.

CHANGE HISTORY
First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

Issue 4
The following change is incorporated in this issue:

• The word “will” is replaced by the word “may” in the ERRORS section.
The `sigaltstack()` function allows a process to define and examine the state of an alternate stack for signal handlers. Signals that have been explicitly declared to execute on the alternate stack will be delivered on the alternate stack.

If `ss` is not a null pointer, it points to a `stack_t` structure that specifies the alternate signal stack that will take effect upon return from `sigaltstack()`. The `ss_flags` member specifies the new stack state. If it is set to SS_DISABLE, the stack is disabled and `ss_sp` and `ss_size` are ignored. Otherwise the stack will be enabled, and the `ss_sp` and `ss_size` members specify the new address and size of the stack.

The range of addresses starting at `ss_sp`, up to but not including `ss_sp + ss_size`, is available to the implementation for use as the stack. This interface makes no assumptions regarding which end is the stack base and in which direction the stack grows as items are pushed.

If `oss` is not a null pointer, on successful completion it will point to a `stack_t` structure that specifies the alternate signal stack that was in effect prior to the call to `sigaltstack()`. The `ss_sp` and `ss_size` members specify the address and size of that stack. The `ss_flags` member specifies the stack’s state, and may contain one of the following values:

- SS_ONSTACK The process is currently executing on the alternate signal stack. Attempts to modify the alternate signal stack while the process is executing on it fails. This flag must not be modified by processes.
- SS_DISABLE The alternate signal stack is currently disabled.

The value SIGSTKSZ is a system default specifying the number of bytes that would be used to cover the usual case when manually allocating an alternate stack area. The value MINSIGSTKSZ is defined to be the minimum stack size for a signal handler. In computing an alternate stack size, a program should add that amount to its stack requirements to allow for the system implementation overhead. The constants SS_ONSTACK, SS_DISABLE, SIGSTKSZ, and MINSIGSTKSZ are defined in `<signal.h>`.

After a successful call to one of the `exec` functions, there are no alternate signal stacks in the new process image.

### RETURN VALUE

Upon successful completion, `sigaltstack()` returns 0. Otherwise, it returns −1 and sets `errno` to indicate the error.

### ERRORS

The `sigaltstack()` function will fail if:

- [EINVAL] The `ss` argument is not a null pointer, and the `ss_flags` member pointed to by `ss` contains flags other than SS_DISABLE.
- [ENOMEM] The size of the alternate stack area is less than MINSIGSTKSZ.
- [EPERM] An attempt was made to modify an active stack.

### APPLICATION USAGE

The following code fragment illustrates a method for allocating memory for an alternate stack:
if ((sigstk.ss_sp = malloc(SIGSTKSZ)) == NULL)
    /* error return */
    sigstk.ss_size = SIGSTKSZ;
    sigstk.ss_flags = 0;
    if (sigaltstack(&sigstk,(stack_t *)0) < 0)
        perror("sigaltstack");

In some implementations, a signal (whether or not indicated to execute on the alternate stack) will always execute on the alternate stack if it is delivered while another signal is being caught using the alternate stack.

On some implementations, stack space is automatically extended as needed. On those implementations, automatic extension is typically not available for an alternate stack. If the stack overflows, the behaviour is undefined.

SEE ALSO
    sigaction(), sigsetjmp(), <signal.h>.

CHANGE HISTORY
    First released in Issue 4, Version 2.
NAME

sigdelset — delete a signal from a signal set

SYNOPSIS

```c
#include <signal.h>

int sigdelset(sigset_t *set, int signo);
```

DESCRIPTION

The `sigdelset()` function deletes the individual signal specified by `signo` from the signal set pointed to by `set`.

RETURN VALUE

Upon successful completion, `sigdelset()` returns 0. Otherwise, it returns −1 and sets `errno` to indicate the error.

ERRORS

The `sigdelset()` function may fail if:

- [EINVAL] The `signo` argument is not a valid signal number, or is an unsupported signal number.

APPLICATION USAGE

Applications should call either `sigemptyset()` or `sigfillset()` at least once for each object of type `sigset_t` prior to any other use of that object. If such an object is not initialised in this way, but is nonetheless supplied as an argument to any of `sigaction()`, `sigaddset()`, `sigdelset()`, `sigismember()`, `sigpending()` or `sigprocmask()`, the results are undefined.

SEE ALSO

- `sigaction()`, `sigaddset()`, `sigemptyset()`, `sigfillset()`, `sigismember()`, `sigpending()`, `sigprocmask()`, `sigsuspend()`, `<signal.h>`.

CHANGE HISTORY

First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

Issue 4

The following change is incorporated in this issue:

- The word “will” is replaced by the word “may” in the `ERRORS` section.
NAME
SIGEMPTYSET — initialise and empty a signal set

SYNOPSIS
#include <signal.h>
int sigemptyset(sigset_t *set);

DESCRIPTION
The sigemptyset() function initialises the signal set pointed to by set, such that all signals defined in this document are excluded.

RETURN VALUE
Upon successful completion, sigemptyset() returns 0. Otherwise, it returns −1 and sets errno to indicate the error.

ERRORS
No errors are defined.

APPLICATION USAGE
Applications should call sigemptyset() or sigfillset() at least once for each object of type sigset_t before any other use of that object.

SEE ALSO
sigaction(), sigaddset(), sigdelset(), sigfillset(), sigismember(), sigpending(), sigprocmask(), sigsuspend(), <signal.h>.

CHANGE HISTORY
First released in Issue 3.
Entry included for alignment with the POSIX.1-1988 standard.
NAME
    sigfillset — initialise and fill a signal set

SYNOPSIS
    #include <signal.h>
    int sigfillset(sigset_t *set);

DESCRIPTION
    The sigfillset() function initialises the signal set pointed to by set, such that all signals defined in this document are included.

RETURN VALUE
    Upon successful completion, sigfillset() returns 0. Otherwise, it returns -1 and sets errno to indicate the error.

ERRORS
    No errors are defined.

APPLICATION USAGE
    Applications should call sigemptyset() or sigfillset() at least once for each object of type sigset_t before any other use of that object.

SEE ALSO
    sigaction(), sigaddset(), sigdelset(), sigemptyset(), sigismember(), sigpending(), sigprocmask(), sigsuspend(), <signal.h>.

CHANGE HISTORY
    First released in Issue 3.
    Entry included for alignment with the POSIX.1-1988 standard.
NAME
sighold, sigignore — add a signal to the signal mask or set a signal disposition to be ignored

SYNOPSIS
#include <signal.h>

int sighold(int sig);
int sigignore(int sig);

DESCRIPTION
Refer to signal().

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME

siginterrupt — allow signals to interrupt functions

SYNOPSIS

UX

#include <signal.h>

int siginterrupt(int sig, int flag);

DESCRIPTION

The siginterrupt() function is used to change the restart behaviour when a function is interrupted by the specified signal. The function siginterrupt(sig, flag) has an effect as if implemented as:

    siginterrupt(int sig, int flag) {
       int ret;
       struct sigaction act;
       (void) sigaction(sig, NULL, &act);
       if (flag)
          act.sa_flags &= ~SA_RESTART;
       else
          act.sa_flags |= SA_RESTART;
       ret = sigaction(sig, &act, NULL);
       return ret;
    }

RETURN VALUE

Upon successful completion, siginterrupt() returns 0. Otherwise −1 is returned and errno is set to indicate the error.

ERRORS

The siginterrupt() function will fail if:

    [EINVAL]   The sig argument is not a valid signal number.

APPLICATION USAGE

The siginterrupt() function supports programs written to historical system interfaces. A portable application, when being written or rewritten, should use sigaction() with the SA_RESTART flag instead of siginterrupt().

SEE ALSO

    sigaction(), <signal.h>.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
sigismember — test for a signal in a signal set

SYNOPSIS
#include <signal.h>

int sigismember(const sigset_t *set, int signo);

DESCRIPTION
The sigismember() function tests whether the signal specified by signo is a member of the set pointed to by set.

RETURN VALUE
Upon successful completion, sigismember() returns 1 if the specified signal is a member of the specified set, or 0 if it is not. Otherwise, it returns −1 and sets errno to indicate the error.

ERRORS
The sigismember() function may fail if:

[EINVAL] The signo argument is not a valid signal number, or is an unsupported signal number.

APPLICATION USAGE
Applications should call either sigemptyset() or sigfillset() at least once for each object of type sigset_t prior to any other use of that object. If such an object is not initialised in this way, but is nonetheless supplied as an argument to any of sigaction(), sigaddset(), sigdelset(), sigismember(), sigpending() or sigprocmask(), the results are undefined.

SEE ALSO
sigaction(), sigaddset(), sigdelset(), sigfillset(), sigemptyset(), sigpending(), sigprocmask(), sigsuspend(), <signal.h>.

CHANGE HISTORY
First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

Issue 4
The following changes are incorporated for alignment with the ISO C standard:

• The type of the argument set is changed from sigset_t* to type const sigset_t*.
• The word “will” is replaced by the word “may” in the ERRORS section.
**NAME**

siglongjmp — non-local goto with signal handling

**SYNOPSIS**

```c
#include <setjmp.h>

void siglongjmp(sigjmp_buf env, int val);
```

**DESCRIPTION**

The `siglongjmp()` function restores the environment saved by the most recent invocation of `sigsetjmp()` in the same process, with the corresponding `sigjmp_buf` argument. If there is no such invocation, or if the function containing the invocation of `sigsetjmp()` has terminated execution in the interim, the behaviour is undefined.

All accessible objects have values as of the time `siglongjmp()` was called, except that the values of objects of automatic storage duration which are local to the function containing the invocation of the corresponding `sigsetjmp()` which do not have volatile-qualified type and which are changed between the `sigsetjmp()` invocation and `siglongjmp()` call are indeterminate.

As it bypasses the usual function call and return mechanisms, `siglongjmp()` will execute correctly in contexts of interrupts, signals and any of their associated functions. However, if `siglongjmp()` is invoked from a nested signal handler (that is, from a function invoked as a result of a signal raised during the handling of another signal), the behaviour is undefined.

The `siglongjmp()` function will restore the saved signal mask if and only if the `env` argument was initialised by a call to `sigsetjmp()` with a non-zero `savemask` argument.

**RETURN VALUE**

After `siglongjmp()` is completed, program execution continues as if the corresponding invocation of `sigsetjmp()` had just returned the value specified by `val`. The `siglongjmp()` function cannot cause `sigsetjmp()` to return 0; if `val` is 0, `sigsetjmp()` returns the value 1.

**ERRORS**

No errors are defined.

**APPLICATION USAGE**

The distinction between `setjmp()` or `longjmp()` and `sigsetjmp()` or `siglongjmp()` is only significant for programs which use `sigaction()`, `sigprocmask()` or `sigsuspend()`.

**SEE ALSO**

`longjmp()`, `setjmp()`, `sigprocmask()`, `sigsetjmp()`, `sigsuspend()`, `<setjmp.h>`.

**CHANGE HISTORY**

First released in Issue 3.

Entry included for alignment with the ISO POSIX-1 standard.

**Issue 4**

The following changes are incorporated in this issue:

- The **APPLICATION USAGE** section is amended.
- An **ERRORS** section is added.
NAME
signal, sigset, sighold, sigrelse, sigignore, sigpause — signal management

SYNOPSIS
#include <signal.h>

void (*signal(int sig, void (*func)(int)))(int);

int sighold(int sig);
int sigignore(int sig);
int sigpause(int sig);
int sigrelse(int sig);

void (*sigset(int sig, void (*disp)(int)))(int);

DESCRIPTION
The signal() function chooses one of three ways in which receipt of the signal number sig is to be subsequently handled. If the value of func is SIG_DFL, default handling for that signal will occur. If the value of func is SIG_IGN, the signal will be ignored. Otherwise, func must point to a function to be called when that signal occurs. Such a function is called a signal handler.

When a signal occurs, if func points to a function, first the equivalent of a:

    signal(sig, SIG_DFL);

is executed or an implementation-dependent blocking of the signal is performed. (If the value of sig is SIGILL, whether the reset to SIG_DFL occurs is implementation-dependent.) Next the equivalent of:

    (*func)(sig);

is executed. The func function may terminate by executing a return statement or by calling abort(), exit(), or longjmp(). If func() executes a return statement and the value of sig was SIGFPE or any other implementation-dependent value corresponding to a computational exception, the behaviour is undefined. Otherwise, the program will resume execution at the point it was interrupted.

If the signal occurs other than as the result of calling abort(), kill() or raise(), the behaviour is undefined if the signal handler calls any function in the standard library other than one of the functions listed on the sigaction() page or refers to any object with static storage duration other than by assigning a value to a static storage duration variable of type volatile sig_atomic_t. Furthermore, if such a call fails, the value of errno is indeterminate.

At program startup, the equivalent of:

    signal(sig, SIG_IGN);

is executed for some signals, and the equivalent of:

    signal(sig, SIG_DFL);

is executed for all other signals (see exec).

The sigset(), sighold(), sigignore(), sigpause() and segrelse() functions provide simplified signal management.

The sigset() function is used to modify signal dispositions. The sig argument specifies the signal, which may be any signal except SIGKILL and SIGSTOP. The disp argument specifies the signal's disposition, which may be SIG_DFL, SIG_IGN or the address of a signal handler. If sigset() is used, and disp is the address of a signal handler, the system will add sig to the calling process'
signal() is used, and disp is equal to SIG_HOLD, sig will be added to the calling process’ signal mask and sig’s disposition will remain unchanged. If sigset() is used, and disp is not equal to SIG_HOLD, sig will be removed from the calling process’ signal mask.

The sighold() function adds sig to the calling process’ signal mask.

The sigrelse() function removes sig from the calling process’ signal mask.

The sigignore() function sets the disposition of sig to SIG_IGN.

The sigpause() function removes sig from the calling process’ signal mask and suspends the calling process until a signal is received.

If the action for the SIGCHLD signal is set to SIG_IGN, child processes of the calling processes will not be transformed into zombie processes when they terminate. If the calling process subsequently waits for its children, and the process has no unwaited for children that were transformed into zombie processes, it will block until all of its children terminate, and wait(), wait3(), waitid() and waitpid() will fail and set errno to [ECHILD].

RETURN VALUE
If the request can be honoured, signal() returns the value of func() for the most recent call to signal() for the specified signal sig. Otherwise, SIG_ERR is returned and a positive value is stored in errno.

UX Upon successful completion, sigset() returns SIG_HOLD if the signal had been blocked and the signal’s previous disposition if it had not been blocked. Otherwise, SIG_ERR is returned and errno is set to indicate the error.

For all other functions, upon successful completion, 0 is returned. Otherwise, -1 is returned and errno is set to indicate the error.

ERRORS
The signal() function will fail if:

[EINVAL] The sig argument is not a valid signal number or an attempt is made to catch a signal that cannot be caught or ignore a signal that cannot be ignored.

The signal() function may fail if:

[EINVAL] An attempt was made to set the action to SIG_DFL for a signal that cannot be caught or ignored (or both).

UX The sigset(), sighold(), sigrelse(), sigignore() and sigpause() functions will fail if:

[EINVAL] The sig argument is an illegal signal number.

The sigset(), and sigignore() functions will fail if:

[EINVAL] An attempt is made to catch a signal that cannot be caught, or to ignore a signal that cannot be ignored.

APPLICATION USAGE
The sigaction() function provides a more comprehensive and reliable mechanism for controlling signals; new applications should use sigaction() rather than signal().

UX The sighold() function, in conjunction with sigrelse() or sigpause(), may be used to establish critical regions of code that require the delivery of a signal to be temporarily deferred.

The sigsuspend() function should be used in preference to sigpause() for broader portability.
SEE ALSO

exec, pause(), sigaction(), waitid(), <signal.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated for alignment with the ISO C standard:

- The function is no longer marked as an extension.
- The argument int is added to the definition of func() in the SYNOPSIS section.
- In Issue 3, this interface cross-referred to sigaction(). This issue provides a complete description of the function as defined in ISO C standard.

Another change is incorporated as follows:

- The APPLICATION USAGE section is added.

Issue 4, Version 2
The following changes are incorporated for X/OPEN UNIX conformance:

- The sighold(), sigignore(), sigpause(), sigrelse() and sigset() functions are added to the SYNOPSIS.
- The DESCRIPTION is updated to describe semantics of the above interfaces.
- Additional text is added to the RETURN VALUE section to describe possible returns from the sigset() function specifically, and all of the above functions in general.
- The ERRORS section is restructured to describe possible error returns from each of the above functions individually.
- The APPLICATION USAGE section is updated to describe certain programming considerations associated with the X/OPEN UNIX functions.
NAME
signgam — storage for sign of \textit{lgamma}()

SYNOPSIS

\begin{verbatim}
#include <math.h>

extern int signgam;
\end{verbatim}

DESCRIPTION
Refer to \textit{lgamma}().

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

\textbf{Issue 4}
The following change is incorporated in this issue:
\begin{itemize}
  \item The header \texttt{<math.h>} is added to the \textbf{SYNOPSIS} section.
\end{itemize}
NAME
   sigpause — remove a signal from the signal mask and suspend the process

SYNOPSIS
   #include <signal.h>
   int sigpause(int sig);

DESCRIPTION
   Refer to signal().

CHANGE HISTORY
   First released in Issue 4, Version 2.
NAME
    sigpending — examine pending signals

SYNOPSIS
    #include <signal.h>
    int sigpending(sigset_t *set);

DESCRIPTION
    The `sigpending()` function stores the set of signals that are blocked from delivery and pending to
the calling process, in the object pointed to by `set`.

RETURN VALUE
    Upon successful completion, `sigpending()` returns 0. Otherwise −1 is returned and `errno` is set to
indicate the error.

ERRORS
    No errors are defined.

SEE ALSO
    `sigaddset()`, `sigdelset()`, `sigemptyset()`, `sigfillset()`, `sigismember()`, `sigprocmask()`, `<signal.h>`.

CHANGE HISTORY
    First released in Issue 3.
NAME

sigprocmask — examine and change blocked signals

SYNOPSIS

#include <signal.h>

int sigprocmask(int how, const sigset_t * set, sigset_t * oset);

DESCRIPTION

The sigprocmask() function allows the calling process to examine and/or change its signal mask.

If the argument set is not a null pointer, it points to a set of signals to be used to change the currently blocked set.

The argument how indicates the way in which the set is changed, and consists of one of the following values:

SIG_BLOCK The resulting set will be the union of the current set and the signal set pointed to by set.

SIG_SETMASK The resulting set will be the signal set pointed to by set.

SIG_UNBLOCK The resulting set will be the intersection of the current set and the complement of the signal set pointed to by set. The resulting set will be the signal set pointed to by set.

If the argument oset is not a null pointer, the previous mask is stored in the location pointed to by oset. If set is a null pointer, the value of the argument how is not significant and the process’ signal mask is unchanged; thus the call can be used to enquire about currently blocked signals.

If there are any pending unblocked signals after the call to sigprocmask(), at least one of those signals will be delivered before the call to sigprocmask() returns.

It is not possible to block those signals which cannot be ignored. This is enforced by the system without causing an error to be indicated.

If any of the SIGFPE, SIGILL or SIGSEGV signals are generated while they are blocked, the result is undefined, unless the signal was generated by a call to kill() or raise().

If sigprocmask() fails, the process’ signal mask is not changed.

RETURN VALUE

Upon successful completion, sigprocmask() returns 0. Otherwise -1 is returned, errno is set to indicate the error and the process’ signal mask will be unchanged.

ERRORS

The sigprocmask() function will fail if:

[EINVAL] The value of the how argument is not equal to one of the defined values.

SEE ALSO

sigaction(), sigaddset(), sigdelset(), sigemptyset(), sigfillset(), sigismember(), sigpending(), sigsuspend(), <signal.h>.

CHANGE HISTORY

First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.
Issue 4

The following change is incorporated for alignment with the ISO POSIX-1 standard:

- The type of the arguments `set` and `oset` are changed from `sigset_t*` to `const sigset_t*`.

Another change is incorporated as follows:

- The **DESCRIPTION** section is changed to indicate that signals can also be generated by `raise()`.
NAME
  sigrelse, sigset — remove a signal from signal mask or modify signal disposition

SYNOPSIS
  #include <signal.h>
  int sigrelse(int sig);
  void (*sigset(int sig, void (*disp)(int)))(int);

DESCRIPTION
  Refer to signal().

CHANGE HISTORY
  First released in Issue 4, Version 2.
NAME

sigsetjmp — set jump point for a non-local goto

SYNOPSIS

```
#include <setjmp.h>

int sigsetjmp(sigjmp_buf env, int savemask);
```

DESCRIPTION

A call to `sigsetjmp()` saves the calling environment in its `env` argument for later use by `siglongjmp()`. It is unspecified whether `sigsetjmp()` is a macro or a function. If a macro definition is suppressed in order to access an actual function, or a program defines an external identifier with the name `sigsetjmp` the behaviour is undefined.

If the value of the `savemask` argument is not 0, `sigsetjmp()` will also save the process’ current signal mask as part of the calling environment.

All accessible objects have values as of the time `siglongjmp()` was called, except that the values of objects of automatic storage duration which are local to the function containing the invocation of the corresponding `sigsetjmp()` which do not have volatile-qualified type and which are changed between the `sigsetjmp()` invocation and `siglongjmp()` call are indeterminate.

An invocation of `sigsetjmp()` must appear in one of the following contexts only:

- the entire controlling expression of a selection or iteration statement
- one operand of a relational or equality operator with the other operand an integral constant expression, with the resulting expression being the entire controlling expression of a selection or iteration statement
- the operand of a unary (!) operator with the resulting expression being the entire controlling expression of a selection or iteration
- the entire expression of an expression statement (possibly cast to `void`).

RETURN VALUE

If the return is from a successful direct invocation, `sigsetjmp()` returns 0. If the return is from a call to `siglongjmp()`, `sigsetjmp()` returns a non-zero value.

ERRORS

No errors are defined.

APPLICATION USAGE

The distinction between `setjmp()`/`longjmp()` and `sigsetjmp()`/`siglongjmp()` is only significant for programs which use `sigaction()`, `sigprocmask()` or `sigsuspend()`.

SEE ALSO

`siglongjmp()`, `signal()`, `sigprocmask()`, `sigsuspend()`, `<setjmp.h>`.

CHANGE HISTORY

First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

Issue 4

The following changes are incorporated in this issue:

- The DESCRIPTION SECTION states that `sigsetjmp()` is a macro or a function. Issue 3 states that it is a macro. Warnings are also added about the suppression of a `sigsetjmp()` macro definition.
• A statement is added to the DESCRIPTION section about the accessibility of objects after a `siglongjmp()` call.

• Text is added to the DESCRIPTION section describing the contexts in which calls to `sigsetjmp()` are valid.
NAME
sigstack — set and/or get alternate signal stack context (TO BE WITHDRAWN)

SYNOPSIS
#include <signal.h>

int sigstack(struct sigstack *ss, struct sigstack *oss);

DESCRIPTION
The sigstack() function allows the calling process to indicate to the system an area of its address
space to be used for processing signals received by the process.

If the ss argument is not a null pointer, it must point to a sigstack structure. The length of the
application-supplied stack must be at least SIGSTKSZ bytes. If the alternate signal stack
overflows, the resulting behaviour is undefined. (See APPLICATION USAGE below.)

• The value of the ss_onstack member indicates whether the process wants the system to use
an alternate signal stack when delivering signals.

• The value of the ss_sp member indicates the desired location of the alternate signal stack
area in the process' address space.

• If the ss argument is a null pointer, the current alternate signal stack context is not changed.

If the oss argument is not a null pointer, it points to a sigstack structure in which the current
alternate signal stack context is placed. The value stored in the ss_onstack member of oss will be
non-zero if the process is currently executing on the alternate signal stack. If the oss argument is
a null pointer, the current alternate signal stack context is not returned.

When a signal's action indicates its handler should execute on the alternate signal stack
(specified by calling sigaction()), the implementation checks to see if the process is currently executing on that stack. If the process is not currently executing on the alternate signal stack, the
system arranges a switch to the alternate signal stack for the duration of the signal handler's
execution.

After a successful call to one of the exec functions, there are no alternate signal stacks in the new
process image.

RETURN VALUE
Upon successful completion, sigstack() returns 0. Otherwise, it returns −1 and sets errno to
indicate the error.

ERRORS
The sigstack() function will fail if:
[EPERM] An attempt was made to modify an active stack.

APPLICATION USAGE
A portable application, when being written or rewritten, should use sigaltstack() instead of
sigstack().

On some implementations, stack space is automatically extended as needed. On those
implementations, automatic extension is typically not available for an alternate stack. If a signal
stack overflows, the resulting behaviour of the process is undefined.

The direction of stack growth is not indicated in the historical definition of struct sigstack. The
only way to portably establish a stack pointer is for the application to determine stack growth
direction, or to allocate a block of storage and set the stack pointer to the middle. The
implementation may assume that the size of the signal stack is SIGSTKSZ as found in
<signal.h>. An implementation that would like to specify a signal stack size other than
SIGSTKSZ should use `sigaltstack()`.

Programs should not use `longjmp()` to leave a signal handler that is running on a stack established with `sigstack()`. Doing so may disable future use of the signal stack. For abnormal exit from a signal handler, `siglongjmp()`, `setcontext()`, or `swapcontext()` may be used. These functions fully support switching from one stack to another.

The `sigstack()` function requires the application to have knowledge of the underlying system's stack architecture. For this reason, `sigaltstack()` is recommended over this function.

SEE ALSO
- `exec`, `fork()`, `_longjmp()`, `longjmp()`, `setjmp()`, `sigaltstack()`, `siglongjmp()`, `sigsetjmp()`, `<signal.h>`.

CHANGE HISTORY
First released in Issue 4, Version 2.
sigsuspend() BASE System Interfaces

NAME
sigsuspend — wait for a signal

SYNOPSIS
#include <signal.h>

int sigsuspend(const sigset_t *sigmask);

DESCRIPTION
The sigsuspend() function replaces the process’ current signal mask with the set of signals pointed to by sigmask and then suspends the process until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process.

If the action is to terminate the process then sigsuspend() will never return. If the action is to execute a signal-catching function, then sigsuspend() will return after the signal-catching function returns, with the signal mask restored to the set that existed prior to the sigsuspend() call.

It is not possible to block signals that cannot be ignored. This is enforced by the system without causing an error to be indicated.

RETURN VALUE
Since sigsuspend() suspends process execution indefinitely, there is no successful completion return value. If a return occurs, –1 is returned and errno is set to indicate the error.

ERRORS
The sigsuspend() function will fail if:

[EINTR] A signal is caught by the calling process and control is returned from the signal-catching function.

SEE ALSO
pause(), sigaction(), sigaddset(), sigdelset(), sigemptyset(), sigfillset(), <signal.h>.

CHANGE HISTORY
First released in Issue 3.
Entry included for alignment with the POSIX.1-1988 standard.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:
• The type of the argument sigmask is changed from sigset_t* to type const sigset_t*.

Another change is incorporated as follows:
• The term “signal handler” is changed to “signal-catching function”.

576 X/Open CAE Specification (1994)
NAME
sin — sine function

SYNOPSIS
#include <math.h>

double sin(double x);

DESCRIPTION
The sin() function computes the sine of its argument x, measured in radians.

RETURN VALUE
Upon successful completion, sin() returns the sine of x.

EX
If x is NaN, NaN is returned and errno may be set to [EDOM].

EX
If x is ±Inf, either 0.0 is returned and errno is set to [EDOM], or NaN is returned and errno may be set to [EDOM].

If the correct result would cause underflow, 0.0 is returned and errno may be set to [ERANGE].

ERRORS
The sin() function may fail if:

EX
[EDOM] The value of x is NaN, or x is ±Inf.

[ERANGE] The result underflows.

EX
No other errors will occur.

APPLICATION USAGE
An application wishing to check for error situations should set errno to 0 before calling sin(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

The sin() function may lose accuracy when its argument is far from 0.0.

SEE ALSO
asin(), isnan(), <math.h>.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

• Removed references to matherr().

• The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the ISO C standard and to rationalise error handling in the mathematics functions.

• The return value specified for [EDOM] is marked as an extension.
NAME
sinh — hyperbolic sine function

SYNOPSIS
#include <math.h>

double sinh(double x);

DESCRIPTION
The sinh() function computes the hyperbolic sine of x.

RETURN VALUE
Upon successful completion, sinh() returns the hyperbolic sine of x.

If the result would cause an overflow, ±HUGE_VAL is returned and errno is set to [ERANGE].
If the result would cause underflow, 0.0 is returned and errno may be set to [ERANGE].

EX
If x is NaN, NaN is returned and errno may be set to [EDOM].

ERRORS
The sinh() function will fail if:
[ERANGE] The result would cause overflow.

The sinh() function may fail if:
[EDOM] The value of x is NaN.
[ERANGE] The result would cause underflow.

EX
No other errors will occur.

APPLICATION USAGE
An application wishing to check for error situations should set errno to 0 before calling sinh(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

SEE ALSO
asinh(), cosh(), isinf(), tanh(), <math.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

- Removed references to matherr().

- The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the ISO C standard and to rationalise error handling in the mathematics functions.

- The return value specified for [EDOM] is marked as an extension.
NAME
sleep — suspend execution for an interval of time

SYNOPSIS
#include <unistd.h>

unsigned int sleep(unsigned int seconds);

DESCRIPTION
The sleep() function will cause the current process to be suspended from execution until either the number of real-time seconds specified by the argument seconds has elapsed or a signal is delivered to the calling process and its action is to invoke a signal-catching function or to terminate the process. The suspension time may be longer than requested due to the scheduling of other activity by the system.

If a SIGALRM signal is generated for the calling process during execution of sleep() and if the SIGALRM signal is being ignored or blocked from delivery, it is unspecified whether sleep() returns when the SIGALRM signal is scheduled. If the signal is being blocked, it is also unspecified whether it remains pending after sleep() returns or it is discarded.

If a SIGALRM signal is generated for the calling process during execution of sleep(), except as a result of a prior call to alarm(), and if the SIGALRM signal is not being ignored or blocked from delivery, it is unspecified whether that signal has any effect other than causing sleep() to return.

If a signal-catching function interrupts sleep() and examines or changes either the time a SIGALRM is scheduled to be generated, the action associated with the SIGALRM signal, or whether the SIGALRM signal is blocked from delivery, the results are unspecified.

If a signal-catching function interrupts sleep() and calls siglongjmp() or longjmp() to restore an environment saved prior to the sleep() call, the action associated with the SIGALRM signal and the time at which a SIGALRM signal is scheduled to be generated are unspecified. It is also unspecified whether the SIGALRM signal is blocked, unless the process’ signal mask is restored as part of the environment.

Interactions between sleep() and any of setitimer(), ualarm() or usleep() are unspecified.

RETURN VALUE
If sleep() returns because the requested time has elapsed, the value returned will be 0. If sleep() returns because of premature arousal due to delivery of a signal, the return value will be the “unslept” amount (the requested time minus the time actually slept) in seconds.

ERRORS
No errors are defined.

SEE ALSO
alarm(), getitimer(), pause(), sigaction(), sigsetjmp(), ualarm(), usleep(), <unistd.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:
• The header <unistd.h> is added to the SYNOPSIS section.
Issue 4, Version 2
The DESCRIPTION is updated to indicate possible interactions with the setitimer(), ualarm() and usleep() functions.
NAME
sprintf — print formatted output

SYNOPSIS
#include <stdio.h>

int sprintf(char *s, const char *format, ...);

DESCRIPTION
Refer to fprintf().

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
• The type of argument format is changed from char * to const char *.

Another change is incorporated as follows:
• The detail for this function is now in fprintf() instead of printf().
NAME
sqrt — square root function

SYNOPSIS
#include <math.h>

double sqrt(double x);

DESCRIPTION
The sqrt() function computes the square root of x, \( \sqrt{x} \).

RETURN VALUE
Upon successful completion, sqrt() returns the square root of x.

EX
If x is NaN, NaN is returned and errno may be set to [EDOM].
EX
If x is negative, 0.0 or NaN is returned and errno is set to [EDOM].

ERRORS
The sqrt() function will fail if:

[EDOM] The value of x is negative.

The sqrt() function may fail if:

EX [EDOM] The value of x is NaN.
EX No other errors will occur.

APPLICATION USAGE
An application wishing to check for error situations should set errno to 0 before calling sqrt(). If
errno is non-zero on return, or the return value is NaN, an error has occurred.

SEE ALSO
isnan(), <math.h>, <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

- Removed references to matherr().
- The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with
  the ISO C standard and to rationalise error handling in the mathematics functions.
- The return value specified for [EDOM] is marked as an extension.
NAME
srand — seed simple pseudo-random number generator

SYNOPSIS
#include <stdlib.h>
void srand(unsigned int seed);

DESCRIPTION
Refer to rand().

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
• The argument seed is explicitly defined as unsigned int.
NAME
srand48 — seed uniformly distributed double-precision pseudo-random number generator

SYNOPSIS

```c
#include <stdlib.h>

void srand48(long int seedval);
```

DESCRIPTION
Refer to `drand48()`.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:
- The header `<stdlib.h>` is added to the SYNOPSIS section.
NAME
srandom — seed pseudorandom number generator

SYNOPSIS

```
#include <stdlib.h>

void srandom(unsigned int seed);
```

DESCRIPTION
Refer to `initstate()`.

CHANGE HISTORY
First released in Issue 4, Version 2.
sscanf()  BASE  System Interfaces

NAME

sscanf — convert formatted input

SYNOPSIS

#include <stdio.h>

int sscanf(const char * s , const char * format , . . .);

DESCRIPTION

Refer to fscanf().

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
- The type of arguments s and format is changed from char * to const char *.

Another change is incorporated as follows:
- The detail for this function is now in fscanf() instead of scanf().
NAME
stat — get file status

SYNOPSIS
OH
#include <sys/types.h>
#include <sys/stat.h>

int stat(const char * path , struct stat * buf );

DESCRIPTION
The stat() function obtains information about the named file and writes it to the area pointed to
by the buf argument. The path argument points to a pathname naming a file. Read, write or
execute permission of the named file is not required, but all directories listed in the pathname
leading to the file must be searchable. An implementation that provides additional or alternate
file access control mechanisms may, under implementation-dependent conditions, cause stat() to fail. In particular, the system may deny the existence of the file specified by path.

The buf argument is a pointer to a stat structure, as defined in the header <sys/stat.h>, into which
information is placed concerning the file.

The stat() function updates any time-related fields (as described in the definition of File Times
Update in the XBD specification), before writing into the stat structure.

The structure members st_mode, st_ino, st_dev, st_uid, st_gid, st_atime, st_ctime and st_mtime will have meaningful values for all file types defined in this document. The value of the member
st_nlink will be set to the number of links to the file.

RETURN VALUE
Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to indicate
the error.

ERRORS
The stat() function will fail if:

[EACCES] Search permission is denied for a component of the path prefix.

UX [EIO] An error occurred while reading from the file system.

UX [ELOOP] Too many symbolic links were encountered in resolving path.

FIPS [ENAMETOOLONG] The length of the path argument exceeds [PATH_MAX] or a pathname
component is longer than [NAME_MAX].

[ENOENT] A component of path does not name an existing file or path is an empty string.
[ENOTDIR] A component of the path prefix is not a directory.

UX The stat() function may fail if:

UX [ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result
whose length exceeds [PATH_MAX].

[EOVERFLOW] A value to be stored would overflow one of the members of the stat structure.

SEE ALSO
tstat(), lstat(), <sys/stat.h>, <sys/types.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- The type of argument path is changed from char * to const char *.
- In the DESCRIPTION section, (a) statements indicating the purpose of this interface and a paragraph defining the contents of stat structure members are added, and (b) the words “extended security controls” are replaced by “additional or alternate file access control mechanisms”.

The following change is incorporated for alignment with the FIPS requirements:

- In the ERRORS section, the condition whereby [ENAMETOOLONG] will be returned if a pathname component is larger than NAME_MAX is now defined as mandatory and marked as an extension.

Another change is incorporated as follows:

- The header <sys/types.h> is now marked as optional (oh); this header need not be included on XSI-conformant systems.

Issue 4, Version 2

The ERRORS section is updated for X/OPEN UNIX conformance as follows:

- In the mandatory section, [EIO] is added to indicate that a physical I/O error has occurred, and [ELOOP] to indicate that too many symbolic links were encountered during pathname resolution.
- In the optional section, a second [ENAMETOOLONG] condition is defined that may report excessive length of an intermediate result of pathname resolution of a symbolic link.
- In the optional section, [EOVERFLOW] is added to indicate that a value to be stored in a member of the stat structure would cause overflow.
NAME
statvfs — get file system information

SYNOPSIS
UX

```c
#include <sys/statvfs.h>

int statvfs(const char *path, struct statvfs *buf);
```

DESCRIPTION
Refer to `fstatvfs()`.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
stderr, stdin, stdout — standard I/O streams

SYNOPSIS
#include <stdio.h>
extern FILE *stderr, *stdin, *stdout;

DESCRIPTION
A file with associated buffering is called a stream and is declared to be a pointer to a defined type FILE. The fopen() function creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. Normally, there are three open streams with constant pointers declared in the <stdio.h> header and associated with the standard open files.

At program startup, three streams are predefined and need not be opened explicitly: standard input (for reading conventional input), standard output (for writing conventional output) and standard error (for writing diagnostic output). When opened, the standard error stream is not fully buffered; the standard input and standard output streams are fully buffered if and only if the stream can be determined not to refer to an interactive device.

The following symbolic values in <unistd.h> define the file descriptors that will be associated with the C-language stdin, stdout and stderr when the application is started:

STDIN_FILENO Standard input value, stdin. Its value is 0.
STDOUT_FILENO Standard output value, stdout. Its value is 1.
STDERR_FILENO Standard error value, stderr. Its value is 2.

SEE ALSO
fclose(), feof(), ferror(), fileno(), fopen(), fread(), fseek(), getc(), gets(), popen(), printf(), putc(), puts(), read(), scanf(), setbuf(), setvbuf(), tmpfile(), ungetc(), vprintf(), <stdio.h>, <unistd.h>.

CHANGE HISTORY
First released in Issue 1.
NAME
step — pattern match with regular expressions (TO BE WITHDRAWN)

SYNOPSIS
EX
#include <regexp.h>

int step(const char *string, const char *expbuf);

DESCRIPTION
Refer to regexp().

CHANGE HISTORY
First released in Issue 2.
Derived from Issue 2 of the SVID.

Issue 4
The following changes are incorporated in this issue:
• The header <regexp.h> is added to the SYNOPSIS section.
• The type of arguments string and expbuf are changed from char * to const char *.
• The interface is marked TO BE WITHDRAWN, because improved functionality is now provided by interfaces introduced for alignment with the ISO POSIX-2 standard.
NAME

strcasecmp, strnケースcmp — case-insensitive string comparisons

SYNOPSIS

#include <strings.h>

int strcasecmp(const char * s1, const char * s2);
int strncasecmp(const char * s1, const char * s2, size_t n);

DESCRIPTION

The `strcasecmp()` function compares, while ignoring differences in case, the string pointed to by `s1` to the string pointed to by `s2`. The `strnケースcmp()` function compares, while ignoring differences in case, not more than `n` bytes from the string pointed to by `s1` to the string pointed to by `s2`.

These functions assume the ASCII character set when equating lower and upper case characters. In the POSIX locale, `strcasecmp()` and `strnケースcmp()` do upper to lower conversions, then a byte comparison. The results are unspecified in other locales.

RETURN VALUE

Upon completion, `strcasecmp()` returns an integer greater than, equal to or less than 0, if the string pointed to by `s1` is, ignoring case, greater than, equal to or less than the string pointed to by `s2` respectively.

Upon successful completion, `strnケースcmp()` returns an integer greater than, equal to or less than 0, if the possibly null-terminated array pointed to by `s1` is, ignoring case, greater than, equal to or less than the possibly null-terminated array pointed to by `s2` respectively.

ERRORS

No errors are defined.

SEE ALSO

<strings.h>.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
strcat — concatenate two strings

SYNOPSIS
#include <string.h>

char *strcat(char *s1, const char *s2);

DESCRIPTION
The strcat() function appends a copy of the string pointed to by s2 (including the terminating
null byte) to the end of the string pointed to by s1. The initial byte of s2 overwrites the null byte
at the end of s1. If copying takes place between objects that overlap, the behaviour is undefined.

RETURN VALUE
The strcat() function returns s1; no return value is reserved to indicate an error.

ERRORS
No errors are defined.

APPLICATION USAGE
This issue is aligned with the ANSI C standard; this does not affect compatibility with XPG3
applications. Reliable error detection by this function was never guaranteed.

SEE ALSO
strncat(), <string.h>.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
• The type of argument s2 is changed from char * to const char *.

Other changes are incorporated as follows:
• The DESCRIPTION section is changed to make it clear that the function manipulates bytes
  rather than (possibly multi-byte) characters.
NAME
strchr — string scanning operation

SYNOPSIS
#include <string.h>
char *strchr(const char *s, int c);

DESCRIPTION
The \texttt{strchr()} function locates the first occurrence of \texttt{c} (converted to an \texttt{unsigned char}) in the string pointed to by \texttt{s}. The terminating null byte is considered to be part of the string.

RETURN VALUE
Upon completion, \texttt{strchr()} returns a pointer to the byte, or a null pointer if the byte was not found.

ERRORS
No errors are defined.

SEE ALSO
\texttt{strrchr()}, \texttt{<string.h>}. 

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
- The type of argument \texttt{s} is changed from \texttt{char *} to \texttt{const char *}.

Other changes are incorporated as follows:
- The \texttt{DESCRIPTION} and \texttt{RETURN VALUE} sections are changed to make it clear that the function manipulates bytes rather than (possibly multi-byte) characters.
- The \texttt{APPLICATION USAGE} section is removed.
NAME
   strcmp — compare two strings

SYNOPSIS
   #include <string.h>
   int strcmp(const char *s1, const char *s2);

DESCRIPTION
   The strcmp() function compares the string pointed to by s1 to the string pointed to by s2.
   The sign of a non-zero return value is determined by the sign of the difference between the
   values of the first pair of bytes (both interpreted as type unsigned char) that differ in the strings
   being compared.

RETURN VALUE
   Upon completion, strcmp() returns an integer greater than, equal to or less than 0, if the string
   pointed to by s1 is greater than, equal to or less than the string pointed to by s2 respectively.

ERRORS
   No errors are defined.

SEE ALSO
   strncmp(), <string.h>.

CHANGE HISTORY
   First released in Issue 1.
   Derived from Issue 1 of the SVID.

Issue 4
   The following change is incorporated for alignment with the ISO C standard:
   • The type of arguments s1 and s2 is changed from char * to const char *.
   Another change is incorporated as follows:
   • The DESCRIPTION section is changed to make it clear that strcmp() compares bytes rather than (possibly multi-byte) characters.
NAME
strcoll — string comparison using collating information

SYNOPSIS
#include <string.h>

int strcoll(const char * s1, const char * s2);

DESCRIPTION
The strcoll() function compares the string pointed to by s1 to the string pointed to by s2, both interpreted as appropriate to the LC_COLLATE category of the current locale.

RETURN VALUE
Upon successful completion, strcoll() returns an integer greater than, equal to or less than 0, according to whether the string pointed to by s1 is greater than, equal to or less than the string pointed to by s2 when both are interpreted as appropriate to the current locale. On error, strcoll() may set errno, but no return value is reserved to indicate an error.

ERRORS
The strcoll() function may fail if:

EX [EINVAL] The s1 or s2 arguments contain characters outside the domain of the collating sequence.

APPLICATION USAGE
Because no return value is reserved to indicate an error, an application wishing to check for error situations should set errno to 0, then call strcoll(), then check errno and if it is non-zero, assume an error has occurred.

This issue is aligned with the ANSI C standard; this does not affect compatibility with XPG3 applications. Reliable error detection by this function was never guaranteed.

The strxfrm() and strcmp() functions should be used for sorting large lists.

SEE ALSO
strcmp(), strxfrm(), <string.h>.

CHANGE HISTORY
First released in Issue 3.

Issue 4
The following changes are incorporated for alignment with the ISO C standard:

• The function is no longer marked as an extension.
• The type of arguments s1 and s2 are changed from char * to const char *.

Other changes are incorporated as follows:

• A paragraph describing how the sign of the return value should be determined is removed from the DESCRIPTION section.
• The [EINVAL] error is marked as an extension.
NAME
strcpy — copy a string

SYNOPSIS
#include <string.h>
char *strcpy(char * s1, const char * s2);

DESCRIPTION
The strcpy() function copies the string pointed to by s2 (including the terminating null byte) into
the array pointed to by s1. If copying takes place between objects that overlap, the behaviour is
undefined.

RETURN VALUE
The strcpy() function returns s1; no return value is reserved to indicate an error.

ERRORS
No errors are defined.

APPLICATION USAGE
Character movement is performed differently in different implementations. Thus overlapping
moves may yield surprises.

This issue is aligned with the ANSI C standard; this does not affect compatibility with XPG3
applications. Reliable error detection by this function was never guaranteed.

SEE ALSO
strncpy(), <string.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
• The type of argument s2 is changed from char * to const char *.

Other changes are incorporated as follows:
• The DESCRIPTION is changed to make it clear that the function manipulates bytes rather
  than (possibly multi-byte) characters.
NAME
strcspn — get length of complementary substring

SYNOPSIS
#include <string.h>
size_t strcspn(const char * s1, const char * s2);

DESCRIPTION
The strcspn() function computes the length of the maximum initial segment of the string pointed to by s1 which consists entirely of bytes not from the string pointed to by s2.

RETURN VALUE
The strcspn() function returns s1; no return value is reserved to indicate an error.

ERRORS
No errors are defined.

SEE ALSO
strspn(), <string.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:

- The type of arguments s1 and s2 is changed from char * to const char *.

Another change is incorporated as follows:

- The DESCRIPTION is changed to make it clear that the function manipulates bytes rather than (possibly multi-byte) characters.
NAME
strdup — duplicate a string

SYNOPSIS
UX
#include <string.h>
char *strdup(const char *s1);

DESCRIPTION
The strdup() function returns a pointer to a new string, which is a duplicate of the string pointed to by s1. The returned pointer can be passed to free(). A null pointer is returned if the new string cannot be created.

RETURN VALUE
The strdup() function returns a pointer to a new string on success. Otherwise it returns a null pointer and sets errno to indicate the error.

ERRORS
The strdup() function may fail if:
[ENOMEM] Storage space available is insufficient.

SEE ALSO
malloc(), free(), <string.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
strerror — get error message string

SYNOPSIS
#include <string.h>
char *strerror(int errnum);

DESCRIPTION
The strerror() function maps the error number in errnum to a locale-dependent error message string and returns a pointer thereto. The string pointed to must not be modified by the program, but may be overwritten by a subsequent call to strerror() or popen().

The contents of the error message strings returned by strerror() should be determined by the setting of the LC_MESSAGES category in the current locale.

The implementation will behave as if no function defined in this document calls strerror().

RETURN VALUE
Upon successful completion, strerror() returns a pointer to the generated message string. On error errno may be set, but no return value is reserved to indicate an error.

ERRORS
The strerror() function may fail if:

- [EINVAL] The value of errnum is not a valid error message number.

APPLICATION USAGE
Because no return value is reserved to indicate an error, an application wishing to check for error situations should set errno to 0, then call strerror(), then check errno and if it is non-zero, assume an error has occurred.

SEE ALSO
<string.h>.

CHANGE HISTORY
First released in Issue 3.

Issue 4
The following change is incorporated for alignment with the ISO C standard:

- The function is no longer marked as an extension.

Other changes are incorporated as follows:

- In the DESCRIPTION section, (a) the term “language-dependent” is replaced by “locale-dependent”, and (b) a statement about the use of the LC_MESSAGES category for determining the language of error messages is added and marked as an extension.

- The fact that strerror() can return a null pointer on failure and set errno is marked as an extension.

- The [EINVAL] error is marked as an extension.

- The FUTURE DIRECTIONS section is removed.
NAME
strfmon — convert monetary value to string

SYNOPSIS

EI  EX
#include <monetary.h>

ssize_t strfmon(char *s, size_t maxsize, const char *format, ...);

DESCRIPTION

The strfmon() function places characters into the array pointed to by s as controlled by the string pointed to by format. No more than maxsize bytes are placed into the array.

The format is a character string that contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in the fetching of zero or more arguments which are converted and formatted. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments are simply ignored.

A conversion specification consists of the following sequence:

- a % character
- optional flags
- optional field width
- optional left precision
- optional right precision
- a required conversion character that determines the conversion to be performed.

Flags

One or more of the following optional flags can be specified to control the conversion:

= An = followed by a single character f which is used as the numeric fill character. The fill character must be representable in a single byte in order to work with precision and width counts. The default numeric fill character is the space character. This flag does not affect field width filling which always uses the space character. This flag is ignored unless a left precision (see below) is specified.

 Specify the style of representing positive and negative currency amounts. Only one of + or ( may be specified. If + is specified, the locale’s equivalent of + and − are used (for example, in the U.S.A.: the empty string if positive and − if negative). If ( is specified, negative amounts are enclosed within parentheses. If neither flag is specified, the + style is used.

 Suppress the currency symbol from the output conversion.

 Specify the alignment. If this flag is present all fields are left-justified (padded to the right) rather than right-justified.
Field Width

$w$ A decimal digit string $w$ specifying a minimum field width in bytes in which the result of the conversion is right-justified (or left-justified if the flag – is specified). The default is 0.

Left Precision

$\#n$ A $\#$ followed by a decimal digit string $n$ specifying a maximum number of digits expected to be formatted to the left of the radix character. This option can be used to keep the formatted output from multiple calls to the `strfmon()` aligned in the same columns. It can also be used to fill unused positions with a special character as in $***123.45$. This option causes an amount to be formatted as if it has the number of digits specified by $n$. If more than $n$ digit positions are required, this conversion specification is ignored. Digit positions in excess of those actually required are filled with the numeric fill character (see the =f flag above).

If grouping has not been suppressed with the `^
` flag, and it is defined for the current locale, grouping separators are inserted before the fill characters (if any) are added. Grouping separators are not applied to fill characters even if the fill character is a digit.

To ensure alignment, any characters appearing before or after the number in the formatted output such as currency or sign symbols are padded as necessary with space characters to make their positive and negative formats an equal length.

Right Precision

.$p$ A period followed by a decimal digit string $p$ specifying the number of digits after the radix character. If the value of the right precision $p$ is 0, no radix character appears. If a right precision is not included, a default specified by the current locale is used. The amount being formatted is rounded to the specified number of digits prior to formatting.

Conversion Characters

The conversion characters and their meanings are:

$i$ The double argument is formatted according to the locale’s international currency format (for example, in the U.S.A.: USD 1,234.56).

$n$ The double argument is formatted according to the locale’s national currency format (for example, in the U.S.A.: $1,234.56$).

% Convert to a %; no argument is converted. The entire conversion specification must be %%.

Locale Information

The LC_MONETARY category of the program’s locale affects the behaviour of this function including the monetary radix character (which may be different from the numeric radix character affected by the LC_NUMERIC category), the grouping separator, the currency symbols and formats. The international currency symbol should be conformant with the ISO 4217: 1987 standard.
RETURN VALUE
If the total number of resulting bytes including the terminating null byte is not more than
maxsize, strfmon() returns the number of bytes placed into the array pointed to by s, not
including the terminating null byte. Otherwise, −1 is returned, the contents of the array are
indeterminate, and errno is set to indicate the error.

ERRORS
The strfmon() function will fail if:
[ENOSYS] The function is not supported.
[E2BIG] Conversion stopped due to lack of space in the buffer.

EXAMPLES
Given a locale for the U.S.A. and the values 123.45, −123.45 and 3456.781:

<table>
<thead>
<tr>
<th>Conversion Specification</th>
<th>Output</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>%n</td>
<td>$123.45</td>
<td>default formatting</td>
</tr>
<tr>
<td></td>
<td>−$123.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$3,456.78</td>
<td></td>
</tr>
<tr>
<td>%11n</td>
<td>$123.45</td>
<td>right align within an 11 character field</td>
</tr>
<tr>
<td></td>
<td>−$123.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$3,456.78</td>
<td></td>
</tr>
<tr>
<td>%#5n</td>
<td>$    123.45</td>
<td>aligned columns for values up to 99,999</td>
</tr>
<tr>
<td></td>
<td>−    123.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$    3,456.78</td>
<td></td>
</tr>
<tr>
<td>%=*#5n</td>
<td>$***123.45</td>
<td>specify a fill character</td>
</tr>
<tr>
<td></td>
<td>−***123.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*3,456.78</td>
<td></td>
</tr>
<tr>
<td>%=0#5n</td>
<td>$000123.45</td>
<td>fill characters do not use grouping</td>
</tr>
<tr>
<td></td>
<td>−$000123.45</td>
<td>even if the fill character is a digit</td>
</tr>
<tr>
<td></td>
<td>$03,456.78</td>
<td></td>
</tr>
<tr>
<td>%ˆ#5n</td>
<td>$    123.45</td>
<td>disable the grouping separator</td>
</tr>
<tr>
<td></td>
<td>−    123.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$    3456.78</td>
<td></td>
</tr>
<tr>
<td>%ˆ#5.0n</td>
<td>$    123</td>
<td>round off to whole units</td>
</tr>
<tr>
<td></td>
<td>−    123</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$    3457</td>
<td></td>
</tr>
<tr>
<td>%ˆ#5.4n</td>
<td>$    123.4500</td>
<td>increase the precision</td>
</tr>
<tr>
<td></td>
<td>−    123.4500</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$    3456.7810</td>
<td></td>
</tr>
<tr>
<td>%(#5n</td>
<td>123.45</td>
<td>use an alternative pos/neg style</td>
</tr>
<tr>
<td></td>
<td>($    123.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$    3,456.78</td>
<td></td>
</tr>
<tr>
<td>%!(#5n</td>
<td>123.45</td>
<td>disable the currency symbol</td>
</tr>
<tr>
<td></td>
<td>(    123.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3,456.78</td>
<td></td>
</tr>
</tbody>
</table>
strfmon()

**FUTURE DIRECTIONS**
This interface is expected to be mandatory in a future issue of this document.

Lower-case conversion characters are reserved for future use and upper-case for implementation-dependent use.

**SEE ALSO**
localeconv(), `<monetary.h>`.

**CHANGE HISTORY**
First released in Issue 4.
NAME
strftime — convert date and time to string

SYNOPSIS
#include <time.h>

size_t strftime(char *s, size_t maxsize, const char *format, const struct tm *timptr);

DESCRIPTION
The strftime() function places bytes into the array pointed to by s as controlled by the string pointed to by format. The format string consists of zero or more conversion specifications and ordinary characters. A conversion specification consists of a % character and a terminating conversion character that determines the conversion specification's behaviour. All ordinary characters (including the terminating null byte) are copied unchanged into the array. If copying takes place between objects that overlap, the behaviour is undefined. No more than maxsize bytes are placed into the array. Each conversion specification is replaced by appropriate characters as described in the following list. The appropriate characters are determined by the program's locale and by the values contained in the structure pointed to by timptr.

Local timezone information is used as though strftime() called tzset().

%a is replaced by the locale's abbreviated weekday name.
%A is replaced by the locale's full weekday name.
%b is replaced by the locale's abbreviated month name.
%B is replaced by the locale's full month name.
%c is replaced by the locale's appropriate date and time representation.
\(\%C\) is replaced by the century number (the year divided by 100 and truncated to an integer) as a decimal number [00-99].
%d is replaced by the day of the month as a decimal number [01,31].
\(\%D\) same as %m/%d/%y.
%e is replaced by the day of the month as a decimal number [1,31]; a single digit is preceded by a space.
%h same as %b.
%H is replaced by the hour (24-hour clock) as a decimal number [00,23].
%I is replaced by the hour (12-hour clock) as a decimal number [01,12].
%j is replaced by the day of the year as a decimal number [001,366].
%m is replaced by the month as a decimal number [01,12].
%M is replaced by the minute as a decimal number [00,59].
%n is replaced by a newline character.
%p is replaced by the locale's equivalent of either a.m. or p.m.
%r is replaced by the time in a.m. and p.m. notation; in the POSIX locale this is equivalent to %I:%M:%S %p.
%S is replaced by the second as a decimal number [00,61].
%T is replaced by the time (%H:%M).
%u is replaced by the weekday as a decimal number [1,7], with 1 representing Monday.
%U is replaced by the week number of the year (Sunday as the first day of the week) as a decimal number [00,53].
%V is replaced by the week number of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is week 53 of the previous year, and the next week is week 1.
%w is replaced by the weekday as a decimal number [0,6], with 0 representing Sunday.
%W is replaced by the week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday are considered to be in week 0.

% x is replaced by the locale’s appropriate date representation.

% X is replaced by the locale’s appropriate time representation.

% y is replaced by the year without century as a decimal number [00,99].

% Y is replaced by the year with century as a decimal number.

% Z is replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists.

%% is replaced by %.

If a conversion specification does not correspond to any of the above, the behaviour is undefined.

**Modified Conversion Specifiers**

Some conversion specifiers can be modified by the E or O modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist for the current locale, (see ERA in the XBD specification, Section 5.3.5) the behaviour will be as if the unmodified conversion specification were used.

-%Ec is replaced by the locale’s alternative appropriate date and time representation.

-%EC is replaced by the name of the base year (period) in the locale’s alternative representation.

-% Ex is replaced by the locale’s alternative date representation.

-% EX is replaced by the locale’s alternative time representation.

-% Ey is replaced by the offset from %EC (year only) in the locale’s alternative representation.

-% EY is replaced by the full alternative year representation.

-% Od is replaced by the day of the month, using the locale’s alternative numeric symbols, filled as needed with leading zeros if there is any alternative symbol for zero, otherwise with leading spaces.

-% Oe is replaced by the day of month, using the locale’s alternative numeric symbols, filled as needed with leading spaces.

-% OH is replaced by the hour (24-hour clock) using the locale’s alternative numeric symbols.

-% Oi is replaced by the hour (12-hour clock) using the locale’s alternative numeric symbols.

-% Om is replaced by the month using the locale’s alternative numeric symbols.

-% OM is replaced by the minutes using the locale’s alternative numeric symbols.

-% OS is replaced by the seconds using the locale’s alternative numeric symbols.

-% Ou is replaced by the weekday as a number in the locale’s alternative representation (Monday=1).

-%OU is replaced by the week number of the year (Sunday as the first day of the week, rules corresponding to %U) using the locale’s alternative numeric symbols.

-%OV is replaced by the week number of the year (Sunday as the first day of the week, rules corresponding to %V) using the locale’s alternative numeric symbols.

-% Ow is replaced by the number of the weekday (Sunday=0) using the locale’s alternative numeric symbols.

-%OW is replaced by the week number of the year (Monday as the first day of the week) using the locale’s alternative numeric symbols.

-%Oy is replaced by the year (offset from %C) in the locale’s alternative representation and using the locale’s alternative symbols.
RETURN VALUE
If the total number of resulting bytes including the terminating null byte is not more than
maxsize, strftime() returns the number of bytes placed into the array pointed to by s, not
including the terminating null byte. Otherwise, 0 is returned and the contents of the array are
indeterminate.

ERRORS
No errors are defined.

APPLICATION USAGE
The range of values for %S is [00,61] rather than [00,59] to allow for the occasional leap second
and even more occasional double leap second.

Some of the conversion specifications marked EX are duplicates of others. They are included for
compatibility with nl_cxtime() and nl_ascxtime(), which were published in Issue 2.

SEE ALSO
asctime(), clock(), ctime(), difftime(), gmtime(), localtime(), mktime(), strftime(), time(), utime(),
<time.h>.

CHANGE HISTORY
First released in Issue 3.

Issue 4
The following changes are incorporated for alignment with the ISO C standard:

• The type of argument format is changed from char * to const char *, and the type of argument
timptr is changed from struct tm* to const struct tm*.

• In the description of the %Z conversion specification, the words “or abbreviation” are added
to indicate that strftime() does not necessarily return a full timezone name.

Other changes are incorporated as follows:

• The DESCRIPTION is expanded to describe modified conversion specifiers.

• %C, %e, %R, %u and %V are added to the list of valid conversion specifications.

• The DESCRIPTION and RETURN VALUE sections are changed to make it clear when the
function uses byte values rather than (possibly multi-byte) character values.
NAME
strlen — get string length

SYNOPSIS
#include <string.h>
size_t strlen(const char *s);

DESCRIPTION
The strlen() function computes the number of bytes in the string to which s points, not including
the terminating null byte.

RETURN VALUE
The strlen() function returns s; no return value is reserved to indicate an error.

ERRORS
No errors are defined.

SEE ALSO
<string.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
• The type of argument s is changed from char * to const char *.

Another change is incorporated as follows:
• The DESCRIPTION section is changed to make it clear that the function works in units of
  bytes rather than (possibly multi-byte) characters.
NAME
strncasecmp — case-insensitive string comparison

SYNOPSIS

```c
#include <strings.h>

int strncasecmp(const char *s1, const char *s2, size_t n);
```

DESCRIPTION
Refer to `strncasecmp()`.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
strncat — concatenate part of two strings

SYNOPSIS
#include <string.h>

char *strncat(char *s1, const char *s2, size_t n);

DESCRIPTION
The strncat() function appends not more than \( n \) bytes (a null byte and bytes that follow it are not appended) from the array pointed to by \( s2 \) to the end of the string pointed to by \( s1 \). The initial byte of \( s2 \) overwrites the null byte at the end of \( s1 \). A terminating null byte is always appended to the result. If copying takes place between objects that overlap, the behaviour is undefined.

RETURN VALUE
The strncat() function returns \( s1 \); no return value is reserved to indicate an error.

ERRORS
No errors are defined.

SEE ALSO
strcat(), <string.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
• The type of argument \( s2 \) is changed from char * to const char *.

Another change is incorporated as follows:
• The DESCRIPTION section is changed to make it clear that the function manipulates bytes rather than (possibly multi-byte) characters.
NAME
strncmp — compare part of two strings

SYNOPSIS
#include <string.h>

int strncmp(const char *s1, const char *s2, size_t n);

DESCRIPTION
The strncmp() function compares not more than n bytes (bytes that follow a null byte are not
compared) from the array pointed to by s1 to the array pointed to by s2.

The sign of a non-zero return value is determined by the sign of the difference between the
values of the first pair of bytes (both interpreted as type unsigned char) that differ in the strings
being compared.

RETURN VALUE
Upon successful completion, strncmp() returns an integer greater than, equal to or less than 0, if
the possibly null-terminated array pointed to by s1 is greater than, equal to or less than the
possibly null-terminated array pointed to by s2 respectively.

ERRORS
No errors are defined.

SEE ALSO
strcmp(), <string.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
• The type of arguments s1 and s2 are changed from char* to const char*.

Another change is incorporated as follows:
• The DESCRIPTION section is changed to make it clear that the function manipulates bytes
rather than (possibly multi-byte) characters.
NAME
strncpy — copy part of a string

SYNOPSIS
#include <string.h>
char *strncpy(char *s1, const char *s2, size_t n);

DESCRIPTION
The strncpy() function copies not more than n bytes (bytes that follow a null byte are not copied) from the array pointed to by s2 to the array pointed to by s1. If copying takes place between objects that overlap, the behaviour is undefined.

If the array pointed to by s2 is a string that is shorter than n bytes, null bytes are appended to the copy in the array pointed to by s1, until n bytes in all are written.

RETURN VALUE
The strncpy() function returns s1; no return value is reserved to indicate an error.

ERRORS
No errors are defined.

APPLICATION USAGE
Character movement is performed differently in different implementations. Thus overlapping moves may yield surprises.

If there is no null byte in the first n bytes of the array pointed to by s2, the result will not be null-terminated.

SEE ALSO
strncpy(), <string.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
• The type of argument s2 is changed from char* to const char*.

Another change is incorporated as follows:
• The DESCRIPTION section is changed to make it clear that the function manipulates bytes rather than (possibly multi-byte) characters.
NAME
   strpbrk — scan string for byte

SYNOPSIS
   #include <string.h>
   char *strpbrk(const char *s1, const char *s2);

DESCRIPTION
   The strpbrk() function locates the first occurrence in the string pointed to by s1 of any byte from the string pointed to by s2.

RETURN VALUE
   Upon successful completion, strpbrk() returns a pointer to the byte or a null pointer if no byte from s2 occurs in s1.

ERRORS
   No errors are defined.

SEE ALSO
   strchr(), strrchr(), <string.h>.

CHANGE HISTORY
   First released in Issue 1.
   Derived from Issue 1 of the SVID.

Issue 4
   The following change is incorporated for alignment with the ISO C standard:
   • The type of arguments s1 and s2 is changed from char * to const char *.

   Another change is incorporated as follows:
   • The DESCRIPTION and RETURN VALUE sections are changed to make it clear that the function works in units of bytes rather than (possibly multi-byte) characters.
NAME
strptime — date and time conversion

SYNOPSIS
#include <time.h>

char *strptime(const char *buf, const char *format, struct tm *tm);

DESCRIPTION
The strptime() function converts the character string pointed to by buf to values which are stored in the tm structure pointed to by tm, using the format specified by format.

The format is composed of zero or more directives. Each directive is composed of one of the following: one or more white-space characters (as specified by isspace()); an ordinary character (neither % nor a white-space character); or a conversion specification. Each conversion specification is composed of a % character followed by a conversion character which specifies the replacement required. There must be white-space or other non-alphanumeric characters between any two conversion specifications. The following conversion specifications are supported:

%a is the day of week, using the locale’s weekday names; either the abbreviated or full name may be specified.
%A is the same as %a.
%b is the month, using the locale’s month names; either the abbreviated or full name may be specified.
%B is the same as %b.
%c is replaced by the locale’s appropriate date and time representation.
%c is replaced by the locale’s appropriate date and time representation.
%C is the century number [0,99]; leading zeros are permitted but not required.
%d is the day of month [1,31]; leading zeros are permitted but not required.
%D is the date as %m/%d/%y.
%e is the same as %d.
%h is the same as %b.
%h is the same as %b.
%H is the hour (24-hour clock) [0,23]; leading zeros are permitted but not required.
%I is the hour (12-hour clock) [1,12]; leading zeros are permitted but not required.
%j is the day number of the year [1,366]; leading zeros are permitted but not required.
%m is the month number [1,12]; leading zeros are permitted but not required.
%M is the minute [0-59]; leading zeros are permitted but not required.
%n is any white space.
%p is the locale’s equivalent of a.m or p.m.
%r is the time as %I:%M:%S .%p.
%R is the time as %H:%M.
%S is the seconds [0,61]; leading zeros are permitted but not required.
%t is any white space.
%T is the time as %H:%M:%S.
%U is the week number of the year (Sunday as the first day of the week) as a decimal number [00,53]; leading zeros are permitted but not required.
%w is the weekday as a decimal number [0,6], with 0 representing Sunday; leading zeros are permitted but not required.
%W is the week number of the year (Monday as the first day of the week) as a decimal number [00,53]; leading zeros are permitted but not required.
%x is the date, using the locale’s date format.
%x is the date, using the locale’s date format.
%X is the time, using the locale’s time format.
%y is the year within the century \([0,99]\); leading zeros are permitted but not required.  
%Y is the year, including the century (for example, 1988).  
%%% is replaced by %.  

**Modified Directives**

Some directives can be modified by the E and O modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified directive. If the alternative format or specification does not exist in the current locale, the behaviour will be as if the unmodified directive were used.

- %Ec is the locale’s alternative appropriate date and time representation.
- %EC is the name of the base year (period) in the locale’s alternative representation.
- %Ex is the locale’s alternative date representation.
- %EX is the locale’s alternative time representation.
- %Ey is the offset from %EC (year only) in the locale’s alternative representation.
- %EY is the full alternative year representation.
- %Od is the day of the month using the locale’s alternative numeric symbols; leading zeros are permitted but not required.
- %Oe is the same as %Od.
- %OH is the hour (24-hour clock) using the locale’s alternative numeric symbols.
- %OI is the hour (12-hour clock) using the locale’s alternative numeric symbols.
- %Om is the month using the locale’s alternative numeric symbols.
- %OM is the minutes using the locale’s alternative numeric symbols.
- %OS is the seconds using the locale’s alternative numeric symbols.
- %OU is the week number of the year (Sunday as the first day of the week) using the locale’s alternative numeric symbols.
- %Ow is the number of the weekday (Sunday=0) using the locale’s alternative numeric symbols.
- %OW is the week number of the year (Monday as the first day of the week) using the locale’s alternative numeric symbols.
- %Oy is the year (offset from %C) in the locale’s alternative representation and using the locale’s alternative numeric symbols.

A directive composed of white-space characters is executed by scanning input up to the first character that is not white space (which remains unscanned), or until no more characters can be scanned.

A directive that is an ordinary character is executed by scanning the next character from the buffer. If the character scanned from the buffer differs from the one comprising the directive, the directive fails, and the differing and subsequent characters remain unscanned.

A series of directives composed of %n, %t, white-space characters or any combination is executed by scanning up to the first character that is not white space (which remains unscanned), or until no more characters can be scanned.

Any other conversion specification is executed by scanning characters until a character matching the next directive is scanned, or until no more characters can be scanned. These characters, except the one matching the next directive, are then compared to the locale values associated with the conversion specifier. If a match is found, values for the appropriate tm structure members are set to values corresponding to the locale information. Case is ignored when matching items in buf such as month or weekday names. If no match is found, `strptime()` fails and no more characters are scanned.
RETURN VALUE
Upon successful completion, `strptime()` returns a pointer to the character following the last character parsed. Otherwise, a null pointer is returned.

ERRORS
The `strptime()` function will fail if:

- [ENOSYS] The functionality is not supported on this implementation.

APPLICATION USAGE
Several “same as” formats, and the special processing of white-space characters are provided in order to ease the use of identical format strings for `strftime()` and `strptime()`.

FUTURE DIRECTIONS
This function is expected to be mandatory in the next issue of this document.

SEE ALSO
`scanf()`, `strftime()`, `time()`, `<time.h>`.

CHANGE HISTORY
First released in Issue 4.
NAME
  
strrchr — string scanning operation

SYNOPSIS

```c
#include <string.h>

char *strrchr(const char *s, int c);
```

DESCRIPTION

The `strrchr()` function locates the last occurrence of `c` (converted to a `char`) in the string pointed to by `s`. The terminating null byte is considered to be part of the string.

RETURN VALUE

Upon successful completion, `strrchr()` returns a pointer to the byte or a null pointer if `c` does not occur in the string.

ERRORS

No errors are defined.

SEE ALSO

`strchr()`, `<string.h>`.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following change is incorporated for alignment with the ISO C standard:

- The type of argument `s` is changed from `char *` to `const char *`.

Another change is incorporated as follows:

- The DESCRIPTION and RETURN VALUE sections are changed to make it clear that the function works in units of bytes rather than (possibly multi-byte) characters.
NAME
strspn — get length of substring

SYNOPSIS
#include <string.h>

size_t strspn(const char *s1, const char *s2);

DESCRIPTION
The strspn() function computes the length of the maximum initial segment of the string pointed to by s1 which consists entirely of bytes from the string pointed to by s2.

RETURN VALUE
The strspn() function returns s1; no return value is reserved to indicate an error.

ERRORS
No errors are defined.

SEE ALSO
strcspn(), <string.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
- The type of arguments s1 and s2 are changed from char * to const char *.

Another change is incorporated as follows:
- The DESCRIPTION section is changed to make it clear that the function works in units of bytes rather than (possibly multi-byte) characters.
NAME
   strstr — find substring

SYNOPSIS
   #include <string.h>
   char *strstr(const char *s1, const char *s2);

DESCRIPTION
   The strstr() function locates the first occurrence in the string pointed to by s1 of the sequence of bytes (excluding the terminating null byte) in the string pointed to by s2.

RETURN VALUE
   Upon successful completion, strstr() returns a pointer to the located string or a null pointer if the string is not found.
   
   If s2 points to a string with zero length, the function returns s1.

ERRORS
   No errors are defined.

SEE ALSO
   strchr(), <string.h>.

CHANGE HISTORY
   First released in Issue 3.
   
   Entry included for alignment with the ANSI C standard.

Issue 4
   The following change is incorporated for alignment with the ISO C standard:
   
   • The type of arguments s1 and s2 are changed from char * to const char *.

   Another change is incorporated as follows:
   
   • The DESCRIPTION section is changed to make it clear that the function works in units of bytes rather than (possibly multi-byte) characters.
NAME
   strtod — convert string to double-precision number

SYNOPSIS
   #include <stdlib.h>
   double strtod(const char *str, char **endptr);

DESCRIPTION
   The `strtod()` function converts the initial portion of the string pointed to by `str` to type `double` representation. First it decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by `isspace()`); a subject sequence interpreted as a floating-point constant; and a final string of one or more unrecognised characters, including the terminating null byte of the input string. Then it attempts to convert the subject sequence to a floating-point number, and returns the result.

   The expected form of the subject sequence is an optional + or − sign, then a non-empty sequence of digits optionally containing a radix character, then an optional exponent part. An exponent part consists of e or E, followed by an optional sign, followed by one or more decimal digits. The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence is empty if the input string is empty or consists entirely of white-space characters, or if the first character that is not white space is other than a sign, a digit or a radix character.

   If the subject sequence has the expected form, the sequence starting with the first digit or the radix character (whichever occurs first) is interpreted as a floating constant of the C language, except that the radix character is used in place of a period, and that if neither an exponent part nor a radix character appears, a radix character is assumed to follow the last digit in the string.

   If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in the object pointed to by `endptr`, provided that `endptr` is not a null pointer.

   The radix character is defined in the program’s locale (category LC_NUMERIC). In the POSIX locale, or in a locale where the radix character is not defined, the radix character defaults to a period (.)

   In other than the POSIX locale, other implementation-dependent subject sequence forms may be accepted.

   If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of `str` is stored in the object pointed to by `endptr`, provided that `endptr` is not a null pointer.

RETURN VALUE
   Upon successful completion, `strtod()` returns the converted value. If no conversion could be performed, 0 is returned, and `errno` may be set to [EINVAL].

   If the correct value is outside the range of representable values, ±HUGE_VAL is returned (according to the sign of the value), and `errno` is set to [ERANGE].

   If the correct value would cause an underflow, 0 is returned and `errno` is set to [ERANGE].
The **strtod()** function will fail if:

- **[ERANGE]** The value to be returned would cause overflow or underflow.

The **strtod()** function may fail if:

- **[EINVAL]** No conversion could be performed.

**APPLICATION USAGE**

Because 0 is returned on error and is also a valid return on success, an application wishing to check for error situations should set *errno* to 0, then call **strtod()**, then check *errno* and if it is non-zero, assume an error has occurred.

**SEE ALSO**

`isspace()`, `localeconv()`, `scanf()`, `setlocale()`, `strtol()`, `<stdlib.h>`, the **XBD** specification, Chapter 5, **Locale**.

**CHANGE HISTORY**

First released in Issue 1.

Derived from Issue 1 of the SVID.

**Issue 4**

The following changes are incorporated for alignment with the ISO C standard:

- The function is no longer marked as an extension.
- The type of argument *str* is changed from `char *` to `const char *`.
- The name of the second argument is changed from *ptr* to *endptr*.
- The precise conditions under which the [ERANGE] error can be set have been defined in the **RETURN VALUE** section.

Other changes are incorporated as follows:

- The **DESCRIPTION** section is changed to make it clear when the function manipulates bytes and when it manipulates characters.
- The [EINVAL] error is added to the **ERRORS** section and marked as an extension.
NAME
   strtok — split string into tokens

SYNOPSIS
   #include <string.h>
   char *strtok(char * s1, const char * s2);

DESCRIPTION
   A sequence of calls to strtok() breaks the string pointed to by s1 into a sequence of tokens, each of which is delimited by a byte from the string pointed to by s2. The first call in the sequence has s1 as its first argument, and is followed by calls with a null pointer as their first argument. The separator string pointed to by s2 may be different from call to call.

   The first call in the sequence searches the string pointed to by s1 for the first byte that is not contained in the current separator string pointed to by s2. If no such byte is found, then there are no tokens in the string pointed to by s1 and strtok() returns a null pointer. If such a byte is found, it is the start of the first token.

   The strtok() function then searches from there for a byte that is contained in the current separator string. If no such byte is found, the current token extends to the end of the string pointed to by s1, and subsequent searches for a token will return a null pointer. If such a byte is found, it is overwritten by a null byte, which terminates the current token. The strtok() function saves a pointer to the following byte, from which the next search for a token will start.

   Each subsequent call, with a null pointer as the value of the first argument, starts searching from the saved pointer and behaves as described above.

   The implementation will behave as if no function defined in this document calls strtok().

RETURN VALUE
   Upon successful completion, strtok() returns a pointer to the first byte of a token. Otherwise, if there is no token, strtok() returns a null pointer.

ERRORS
   No errors are defined.

SEE ALSO
   <string.h>.

CHANGE HISTORY
   First released in Issue 1.
   Derived from Issue 1 of the SVID.

   Issue 4
   The following changes are incorporated for alignment with the ISO C standard:
   • The function is no longer marked as an extension.
   • The type of argument s2 is changed from char * to const char *.

   Another change is incorporated as follows:
   • The DESCRIPTION section is changed to make it clear that the function manipulates bytes rather than (possibly multi-byte) characters.
NAME
strtol — convert string to long integer

SYNOPSIS
#include <stdlib.h>

long int strtol(const char * str, char **endptr, int base);

DESCRIPTION
The `strtol()` function converts the initial portion of the string pointed to by `str` to a type `long int`
representation. First it decomposes the input string into three parts: an initial, possibly empty,
sequence of white-space characters (as specified by `isspace()`); a subject sequence interpreted as
an integer represented in some radix determined by the value of `base`; and a final string of one or
more unrecognised characters, including the terminating null byte of the input string. Then it
attempts to convert the subject sequence to an integer, and returns the result.

If the value of `base` is 0, the expected form of the subject sequence is that of a decimal constant,
octal constant or hexadecimal constant, any of which may be preceded by a + or − sign. A
decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An
octal constant consists of the prefix 0 optionally followed by a sequence of the digits 0 to 7 only.
a hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal
digits and letters a (or A) to f (or F) with values 10 to 15 respectively.

If the value of `base` is between 2 and 36, the expected form of the subject sequence is a sequence
of letters and digits representing an integer with the radix specified by `base`, optionally preceded
by a + or − sign. The letters from a (or A) to z (or Z) inclusive are ascribed the values 10 to 35;
only letters whose ascribed values are less than that of `base` are permitted. If the value of `base` is
16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following
the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting
with the first non-white-space character, that is of the expected form. The subject sequence
contains no characters if the input string is empty or consists entirely of white-space characters,
or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of `base` is 0, the sequence of
characters starting with the first digit is interpreted as an integer constant. If the subject
sequence has the expected form and the value of `base` is between 2 and 36, it is used as the base
for conversion, ascribing to each letter its value as given above. If the subject sequence begins
with a minus sign, the value resulting from the conversion is negated. A pointer to the final
string is stored in the object pointed to by `endptr`, provided that `endptr` is not a null pointer.

In other than the POSIX locale, additional implementation-dependent subject sequence forms
may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed;
the value of `str` is stored in the object pointed to by `endptr`, provided that `endptr` is not a null pointer.

RETURN VALUE
Upon successful completion `strtol()` returns the converted value, if any. If no conversion could
be performed, 0 is returned and `errno` may be set to [EINVAL].

If the correct value is outside the range of representable values, LONG_MAX or LONG_MIN is
returned (according to the sign of the value), and `errno` is set to [ERANGE].
**ERRORS**

The `strtol()` function will fail if:

- **[ERANGE]** The value to be returned is not representable.

The `strtol()` function may fail if:

- **[EINVAL]** The value of `base` is not supported.

**APPLICATION USAGE**

Because 0, LONG_MIN and LONG_MAX are returned on error and are also valid returns on success, an application wishing to check for error situations should set `errno` to 0, then call `strtol()`, then check `errno` and if it is non-zero, assume an error has occurred.

**SEE ALSO**

`isalpha()`, `scanf()`, `strtol()`, `<stdlib.h>`.

**CHANGE HISTORY**

First released in Issue 1.

Derived from Issue 1 of the SVID.

**Issue 4**

The following changes are incorporated for alignment with the ISO C standard:

- The function is no longer marked as an extension.
- The type of argument `str` is changed from `char *` to `const char *`.
- The name of the second argument is changed from `ptr` to `endptr`.
- The `DESCRIPTION` section is changed to indicate permitted forms of the subject sequence when `base` is 0.
- The `RETURN VALUE` section is changed to indicate that LONG_MAX or LONG_MIN will be returned if the converted value is too large or too small.

Other changes are incorporated as follows:

- The `DESCRIPTION` section is changed to make it clear when the function manipulates bytes and when it manipulates characters.
- In the `RETURN VALUE` section, text indicating that `errno` will be set when 0 is returned is marked as an extension.
- The `ERRORS` section is updated in line with the `RETURN VALUE` section.
NAME
strtoul — convert string to unsigned long

SYNOPSIS
#include <stdlib.h>

unsigned long int strtoul(const char *str, char **endptr, int base);

DESCRIPTION
The *strtol*() function converts the initial portion of the string pointed to by *str* to a type
*unsigned long int* representation. First it decomposes the input string into three parts: an initial,
possibly empty, sequence of white-space characters (as specified by *isspace*()); a subject sequence
interpreted as an integer represented in some radix determined by the value of *base*; and a final
string of one or more unrecognised characters, including the terminating null byte of the input
string. Then it attempts to convert the subject sequence to an unsigned integer, and returns the
result.

If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant,
octal constant or hexadecimal constant, any of which may be preceded by a + or − sign. A
decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An
octal constant consists of the prefix 0 optionally followed by a sequence of the digits 0 to 7 only.
A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal
digits and letters a (or A) to f (or F) with values 10 to 15 respectively.

If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence
of letters and digits representing an integer with the radix specified by *base*, optionally preceded
by a + or − sign. The letters from a (or A) to z (or Z) inclusive are ascribed the values 10 to 35;
only letters whose ascribed values are less than that of *base* are permitted. If the value of *base* is
16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following
the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting
with the first non-white-space character, that is of the expected form. The subject sequence
contains no characters if the input string is empty or consists entirely of white-space characters,
or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of *base* is 0, the sequence of
characters starting with the first digit is interpreted as an integer constant. If the subject
sequence has the expected form and the value of *base* is between 2 and 36, it is used as the base
for conversion, ascribing to each letter its value as given above. If the subject sequence begins
with a minus sign, the value resulting from the conversion is negated. A pointer to the final
string is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

In other than the POSIX locale, additional implementation-dependent subject sequence forms
may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed;
the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

RETURN VALUE
Upon successful completion *strtol*() returns the converted value, if any. If no conversion could
be performed, 0 is returned and *errno* may be set to [EINVAL]. If the correct value is outside the
range of representable values, {ULONG_MAX} is returned and *errno* is set to [ERANGE].

System Interfaces and Headers Issue 4, Version 2 625
ERRORS

The `strtoul()` function will fail if:

EX [EINVAL] The value of `base` is not supported.

[ERANGE] The value to be returned is not representable.

The `strtoul()` function may fail if:

EX [EINVAL] No conversion could be performed.

APPLICATION USAGE

Because 0 and `ULONG_MAX` are returned on error and are also valid returns on success, an application wishing to check for error situations should set `errno` to 0, then call `strtoul()`, then check `errno` and if it is non-zero, assume an error has occurred.

Unlike `strtod()` and `strtol()`, `strtoul()` must always return a non-negative number; so, using the return value of `strtoul()` for out-of-range numbers with `strtoul()` could cause more severe problems than just loss of precision if those numbers can ever be negative.

SEE ALSO

`isalpha()`, `scanf()`, `strtod()`, `strtol()`, `<stdlib.h>`.

CHANGE HISTORY

First released in Issue 4.

Derived from the ANSI C standard.
NAME
strxfrm — string transformation

SYNOPSIS
#include <string.h>
size_t strxfrm(char * s1, const char * s2, size_t n);

DESCRIPTION
The strxfrm() function transforms the string pointed to by s2 and places the resulting string into
the array pointed to by s1. The transformation is such that if strcmp() is applied to two
transformed strings, it returns a value greater than, equal to or less than 0, corresponding to the
result of strcoll() applied to the same two original strings. No more than n bytes are placed into
the resulting array pointed to by s1, including the terminating null byte. If n is 0, s1 is permitted
to be a null pointer. If copying takes place between objects that overlap, the behaviour is
undefined.

RETURN VALUE
Upon successful completion, strxfrm() returns the length of the transformed string (not
including the terminating null byte). If the value returned is n or more, the contents of the array
pointed to by s1 are indeterminate.

EX On error strxfrm() may set errno but no return value is reserved to indicate an error.

ERRORS
The strxfrm() function may fail if:

EX [EINVAL] The string pointed to by the s2 argument contains characters outside the
domain of the collating sequence.

APPLICATION USAGE
The transformation function is such that two transformed strings can be ordered by strcmp() as
appropriate to collating sequence information in the program's locale (category LC_COLLATE).
The fact that when n is 0, s1 is permitted to be a null pointer, is useful to determine the size of the
s1 array prior to making the transformation.
Because no return value is reserved to indicate an error, an application wishing to check for error
situations should set errno to 0, then call strcoll(), then check errno and if it is non-zero, assume
an error has occurred.
This issue is aligned with the ANSI C standard; this does not affect compatibility with XPG3
applications. Reliable error detection by this function was never guaranteed.

SEE ALSO
strcmp(), strcoll(), <string.h>.

CHANGE HISTORY
First released in Issue 3.
Enter included for alignment with the ISO C standard.

Issue 4
The following changes are incorporated for alignment with the ISO C standard:
  • The function is no longer marked as an extension.
  • The type of argument s2 is changed from char * to const char *.
Other changes are incorporated as follows:

- The DESCRIPTION section is changed to make it clear when the function manipulates byte values and when it manipulates characters.

- The sentence describing error returns in the RETURN VALUE section is marked as an extension, as is the [EINVAL] error.

- The APPLICATION USAGE section is expanded.
NAME
swab — swap bytes

SYNOPSIS
EX
```c
#include <unistd.h>

void swab(const void * src, void * dest, ssize_t nbytes);
```

DESCRIPTION
The `swab()` function copies `nbytes` bytes, which are pointed to by `src`, to the object pointed to by `dest`, exchanging adjacent bytes. The `nbytes` argument should be even. If `nbytes` is odd `swab()` copies and exchanges `nbytes−1` bytes and the disposition of the last byte is unspecified. If copying takes place between objects that overlap, the behaviour is undefined. If `nbytes` is negative, `swab()` does nothing.

ERRORS
No errors are defined.

SEE ALSO
<unistd.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:
- The header `<unistd.h>` is added to the SYNOPSIS section.
- The type of argument `src` is changed from `char *` to `const void*`, `dest` is changed from `char *` to `void*`, and `nbytes` is changed from `int` to `ssize_t`.
- The DESCRIPTION section now states explicitly that copying between overlapping objects results in undefined behaviour. is changed to take account of the type change to `nbyte`; that is, previously it was defined as `int` and could be positive or negative, whereas now it is defined as an `unsigned` type. Also a statement about overlapping objects is added to the DESCRIPTION section.
- The APPLICATION USAGE section is removed.
NAME
swapcontext — swap user context

SYNOPSIS
#include <ucontext.h>

int swapcontext(ucontext_t *oucp, const ucontext_t *ucp);

DESCRIPTION
Refer to makecontext().

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
symlink — make symbolic link to a file

SYNOPSIS
UX
#include <unistd.h>
int symlink(const char *path1, const char *path2);

DESCRIPTION
The symlink() function creates a symbolic link. Its name is the pathname pointed to by path2, which must be a pathname that does not name an existing file or symbolic link. The contents of the symbolic link are the string pointed to by path1.

RETURN VALUE
Upon successful completion, symlink() returns 0. Otherwise, it returns −1 and sets errno to indicate the error.

ERRORS
The symlink() function will fail if:

[EACCES] Write permission is denied in the directory where the symbolic link is being created, or search permission is denied for a component of the path prefix of path2.

[EEXIST] The path2 argument names an existing file or symbolic link.

[EIO] An I/O error occurs while reading from or writing to the file system.

[ELOOP] Too many symbolic links were encountered in resolving path2.

[ENAMETOOLONG] The length of the path2 argument exceeds {PATH_MAX}, or a pathname component is longer than {NAME_MAX}.

[ENOENT] A component of path2 does not name an existing file or path2 is an empty string.

[ENOSPC] The directory in which the entry for the new symbolic link is being placed cannot be extended because no space is left on the file system containing the directory, or the new symbolic link cannot be created because no space is left on the file system which will contain the link, or the file system is out of file-allocation resources.

[ENOTDIR] A component of the path prefix of path2 is not a directory.

[EROFS] The new symbolic link would reside on a read-only file system.

The symlink() function may fail if:

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.

APPLICATION USAGE
Like a hard link, a symbolic link allows a file to have multiple logical names. The presence of a hard link guarantees the existence of a file, even after the original name has been removed. A symbolic link provides no such assurance; in fact, the file named by the path1 argument need not exist when the link is created. A symbolic link can cross file system boundaries.

Normal permission checks are made on each component of the symbolic link pathname during its resolution.
SEE ALSO

lchown(), link(), lstat(), open(), readlink(), <unistd.h>.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
sync — schedule filesystem updates

SYNOPSIS
UX
#include <unistd.h>

void sync(void);

DESCRIPTION
The sync() function causes all information in memory that updates file systems to be scheduled for writing out to all file systems.

The writing, although scheduled, is not necessarily complete upon return from sync().

RETURN VALUE
The sync() function returns no value.

ERRORS
No errors are defined.

SEE ALSO
fsync(), <unistd.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
sysconf — get configurable system variables

SYNOPSIS
#include <unistd.h>

long int sysconf(int name);

DESCRIPTION
The sysconf() function provides a method for the application to determine the current value of a configurable system limit or option (variable).

The name argument represents the system variable to be queried. The following table lists the minimal set of system variables from <limits.h>, <unistd.h> or <time.h> (for CLK_TCK) that can be returned by sysconf(), and the symbolic constants, defined in <unistd.h> that are the corresponding values used for name:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value of name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARG_MAX</td>
<td>_SC_ARG_MAX</td>
</tr>
<tr>
<td>BC_BASE_MAX</td>
<td>_SC_BC_BASE_MAX</td>
</tr>
<tr>
<td>BC_DIM_MAX</td>
<td>_SC_BC_DIM_MAX</td>
</tr>
<tr>
<td>BC_SCALE_MAX</td>
<td>_SC_BC_SCALE_MAX</td>
</tr>
<tr>
<td>BC_STRING_MAX</td>
<td>_SC_BC_STRING_MAX</td>
</tr>
<tr>
<td>CHILD_MAX</td>
<td>_SC_CHILD_MAX</td>
</tr>
<tr>
<td>CLK_TCK</td>
<td>_SC_CLK_TCK</td>
</tr>
<tr>
<td>COLL_WEIGHTS_MAX</td>
<td>_SC_COLL_WEIGHTS_MAX</td>
</tr>
<tr>
<td>EXPR_NEST_MAX</td>
<td>_SC_EXPR_NEST_MAX</td>
</tr>
<tr>
<td>LINE_MAX</td>
<td>_SC_LINE_MAX</td>
</tr>
<tr>
<td>NGROUPS_MAX</td>
<td>_SC_NGROUPS_MAX</td>
</tr>
<tr>
<td>OPEN_MAX</td>
<td>_SC_OPEN_MAX</td>
</tr>
<tr>
<td>PASS_MAX</td>
<td>_SC_PASS_MAX (TO BE WITHDRAWN)</td>
</tr>
<tr>
<td>_POSIX2_C_BIND</td>
<td>_SC_2_C_BIND</td>
</tr>
<tr>
<td>_POSIX2_C_DEV</td>
<td>_SC_2_C_DEV</td>
</tr>
<tr>
<td>_POSIX2_C_VERSION</td>
<td>_SC_2_C_VERSION</td>
</tr>
<tr>
<td>_POSIX2_CHAR_TERM</td>
<td>_SC_2_CHAR_TERM</td>
</tr>
<tr>
<td>_POSIX2_FORT_DEV</td>
<td>_SC_2_FORT_DEV</td>
</tr>
<tr>
<td>_POSIX2_FORT_RUN</td>
<td>_SC_2_FORT_RUN</td>
</tr>
<tr>
<td>_POSIX2_LOCALEDEF</td>
<td>_SC_2_LOCALEDEF</td>
</tr>
<tr>
<td>_POSIX2_SW_DEV</td>
<td>_SC_2_SW_DEV</td>
</tr>
<tr>
<td>_POSIX2_UPE</td>
<td>_SC_2_UPE</td>
</tr>
<tr>
<td>_POSIX2_VERSION</td>
<td>_SC_2_VERSION</td>
</tr>
<tr>
<td>_POSIX_JOB_CONTROL</td>
<td>_SC_JOB_CONTROL</td>
</tr>
<tr>
<td>_POSIX_SAVED_IDS</td>
<td>_SC_SAVED_IDS</td>
</tr>
<tr>
<td>_POSIX_VERSION</td>
<td>_SC_VERSION</td>
</tr>
<tr>
<td>RE_DUP_MAX</td>
<td>_SC_RE_DUP_MAX</td>
</tr>
<tr>
<td>STREAM_MAX</td>
<td>_SC_STREAM_MAX</td>
</tr>
<tr>
<td>TZNAME_MAX</td>
<td>_SC_TZNAME_MAX</td>
</tr>
</tbody>
</table>
# System Interfaces

## sysconf()

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value of name</th>
</tr>
</thead>
<tbody>
<tr>
<td>_XOPEN_CRYPT</td>
<td>_SC_XOPEN_CRYPT</td>
</tr>
<tr>
<td>_XOPEN_ENH_I18N</td>
<td>_SC_XOPEN_ENH_I18N</td>
</tr>
<tr>
<td>_XOPEN_SHM</td>
<td>_SC_XOPEN_SHM</td>
</tr>
<tr>
<td>_XOPEN_VERSION</td>
<td>_SC_XOPEN_VERSION</td>
</tr>
<tr>
<td>_XOPEN_XCU_VERSION</td>
<td>_SC_XOPEN_XCU_VERSION</td>
</tr>
<tr>
<td>ATEXIT_MAX</td>
<td>_SC_ATEXIT_MAX</td>
</tr>
<tr>
<td>IOV_MAX</td>
<td>_SC_IOV_MAX</td>
</tr>
<tr>
<td>PAGESIZE</td>
<td>_SC_PAGESIZE</td>
</tr>
<tr>
<td>PAGESIZE</td>
<td>_SC_PAGE_SIZE</td>
</tr>
<tr>
<td>_XOPEN_UNIX</td>
<td>_SC_XOPEN_UNIX</td>
</tr>
</tbody>
</table>

## RETURN VALUE

If `name` is an invalid value, `sysconf()` returns −1 and sets `errno` to indicate the error. If the variable corresponding to `name` is associated with functionality that is not supported by the system, `sysconf()` returns −1 without changing the value of `errno`.

Otherwise, `sysconf()` returns the current variable value on the system. The value returned will not be more restrictive than the corresponding value described to the application when it was compiled with the implementation's `<limits.h>`, `<unistd.h>` or `<time.h>`. The value will not change during the lifetime of the calling process.

## ERRORS

The `sysconf()` function will fail if:

- [EINVAL] The value of the `name` argument is invalid.

## APPLICATION USAGE

As −1 is a permissible return value in a successful situation, an application wishing to check for error situations should set `errno` to 0, then call `sysconf()`, and, if it returns −1, check to see if `errno` is non-zero.

If the value of:

```
sysconf (_SC_2_VERSION)
```

is not equal to the value of the `_POSIX2_VERSION` symbolic constant, the utilities available via `system()` or `popen()` might not behave as described in the XCU specification. This would mean that the application is not running in an environment that conforms to the XCU specification. Some applications might be able to deal with this, others might not. However, the interfaces defined in this document will continue to operate as specified, even if:

```
sysconf (_SC_2_VERSION)
```

reports that the utilities no longer perform as specified.

## SEE ALSO

`pathconf()`, `<limits.h>`, `<time.h>`, `<unistd.h>`, `<unistd.h>`.

## CHANGE HISTORY

First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

### Issue 4

The following change is incorporated for alignment with the ISO POSIX-1 standard:

...
The variables \{STREAM_MAX\} and \{TZNAME_MAX\} are added to the table of variables in the **DESCRIPTION** section.

The following change is incorporated for alignment with the ISO POSIX-2 standard:

- The following variables are added to the table of configurable system limits in the **DESCRIPTION** section:
  
<table>
<thead>
<tr>
<th><strong>Variable</strong></th>
<th><strong>System Limit</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>BC_BASE_MAX</td>
<td>_POSIX2_C_BIND _POSIX2_SW_DEV</td>
</tr>
<tr>
<td>BC_DIM_MAX</td>
<td>_POSIX2_C_DEV _POSIX2_VERSION</td>
</tr>
<tr>
<td>BC_SCALE_MAX</td>
<td>_POSIX2_C_VERSION RE_DUP_MAX</td>
</tr>
<tr>
<td>BC_STRING_MAX</td>
<td>_POSIX2_CHAR_TERM</td>
</tr>
<tr>
<td>COLL_WEIGHTS_MAX</td>
<td>_POSIX2_FORT_DEV</td>
</tr>
<tr>
<td>EXPR_NEST_MAX</td>
<td>_POSIX2_FORT_RUN</td>
</tr>
<tr>
<td>LINE_MAX</td>
<td>_POSIX2_LOCALEDEF</td>
</tr>
</tbody>
</table>

Other changes are incorporated as follows:

- The type of the function return value is expanded to `long int`.
- `_XOPEN_VERSION` is added to the table of configurable system limits; this should have been included in Issue 3.
- The following variables are added to the table of configurable system limits in the **DESCRIPTION** section and marked as extensions:
  
<table>
<thead>
<tr>
<th><strong>Extension</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>_XOPEN_CRYPT _XOPEN_ENH_I18N _XOPEN_SHM</td>
</tr>
<tr>
<td>_XOPEN_UNIX</td>
</tr>
</tbody>
</table>

- In the **RETURN VALUE** section the header `<time.h>` is given as an alternative to `<limits.h>` and `<unistd.h>`.
- The second paragraph is added to the **APPLICATION USAGE** section.

### Issue 4, Version 2

For X/OPEN UNIX conformance, the ATEXIT_MAX, IOV_MAX, PAGESIZE, PAGE_SIZE and _XOPEN_UNIX variables are added to the list of configurable system values that can be determined by calling `sysconf()`.
**NAME**

syslog — log a message

**SYNOPSIS**

```
#include <syslog.h>

void syslog(int priority, const char *message, ... /* argument */);
```

**DESCRIPTION**

Refer to `closelog()`.

**CHANGE HISTORY**

First released in Issue 4, Version 2.
NAME
system — issue a command

SYNOPSIS
#include <stdlib.h>

int system(const char *command);

DESCRIPTION
The system() function passes the string pointed to by command to the host environment to be executed by a command processor in an implementation-dependent manner. If the implementation supports the XCU specification commands, the environment of the executed command will be as if a child process were created using fork(), and the child process invoked the sh utility (see sh in the XCU specification) using execl() as follows:

   execl(<shell path>, "sh", "-c", command, (char *)0);

where <shell path> is an unspecified pathname for the sh utility.

The system() function ignores the SIGINT and SIGQUIT signals, and blocks the SIGCHLD signal, while waiting for the command to terminate. If this might cause the application to miss a signal that would have killed it, then the application should examine the return value from system() and take whatever action is appropriate to the application if the command terminated due to receipt of a signal.

The system() function will not affect the termination status of any child of the calling processes other than the process or processes it itself creates.

The system() function will not return until the child process has terminated.

RETURN VALUE
If command is a null pointer, system() returns non-zero only if a command processor is available.

If command is not a null pointer, system() returns the termination status of the command language interpreter in the format specified by waitpid(). The termination status of the command language interpreter is as specified for the sh utility, except that if some error prevents the command language interpreter from executing after the child process is created, the return value from system() will be as if the command language interpreter had terminated using exit(127) or _exit(127). If a child process cannot be created, or if the termination status for the command language interpreter cannot be obtained, system() returns −1 and sets errno to indicate the error.

ERRORS
The system() function may set errno values as described by fork().

In addition, system() may fail if:

ECHILD    The status of the child process created by system() is no longer available.

APPLICATION USAGE
If the return value of system() is not −1, its value can be decoded through the use of the macros described in <sys/wait.h>. For convenience, these macros are also provided in <stdlib.h>.

To determine whether or not the XCU specification’s environment is present, use:

sysconf(_SC_2_VERSION)

The sh may not be available after a call to chroot().
Note that, while `system()` must ignore SIGINT and SIGQUIT and block SIGCHLD while waiting for the child to terminate, the handling of signals in the executed command is as specified by `fork()` and `exec`. For example, if SIGINT is being caught or is set to SIG_DFL when `system()` is called, then the child will be started with SIGINT handling set to SIG_DFL.

Ignoring SIGINT and SIGQUIT in the parent process prevents coordination problems (two processes reading from the same terminal, for example) when the executed command ignores or catches one of the signals. It is also usually the correct action when the user has given a command to the application to be executed synchronously (as in the "!" command in many interactive applications). In either case, the signal should be delivered only to the child process, not to the application itself. There is one situation where ignoring the signals might have less than the desired effect. This is when the application uses `system()` to perform some task invisible to the user. If the user typed the interrupt character (\^C, for example) while `system()` is being used in this way, one would expect the application to be killed, but only the executed command will be killed. Applications that use `system()` in this way should carefully check the return status from `system()` to see if the executed command was successful, and should take appropriate action when the command fails.

Blocking SIGCHLD while waiting for the child to terminate prevents the application from catching the signal and obtaining status from `system()`’s child process before `system()` can get the status itself.

The context in which the utility is ultimately executed may differ from that in which `system()` was called. For example, file descriptors that have the FD_CLOEXEC flag set will be closed, and the process ID and parent process ID will be different. Also, if the executed utility changes its environment variables or its current working directory, that change will not be reflected in the caller's context.

There is no defined way for an application to find the specific path for the shell. However, `confstr()` can provide a value for PATH that is guaranteed to find the sh utility.

SEE ALSO
`exec`, `pipe()`, `waitpid()`, `<limits.h>`, `<signal.h>`, `<stdlib.h>`, the XCU specification.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated for alignment with the ISO POSIX-2 standard:

- The function is no longer marked as an extension.
- The name of the argument is changed from string to command, and its type is changed from char * to const char *.
- The DESCRIPTION and RETURN VALUE sections are completely replaced to bring them in line with ISO POSIX-2 standard. They still describe essentially the same functionality, albeit that the definition is more complete.
- The ERRORS section is changed to indicate that `system()` may return error values described for `fork()`.
- The APPLICATION USAGE section is added.
NAME
  tan — tangent function

SYNOPSIS
  #include <math.h>
  double tan(double x);

DESCRIPTION
  The tan() function computes the tangent of its argument x, measured in radians.

RETURN VALUE
  Upon successful completion, tan() returns the tangent of x.

EX
  If x is NaN, NaN is returned and errno may be set to [EDOM].
EX
  If x is ±Inf, either 0.0 is returned and errno is set to [EDOM], or NaN is returned and errno may be set to [EDOM].

  If the correct value would cause overflow, ±HUGE_VAL is returned and errno is set to [ERANGE].
  If the correct value would cause underflow, 0.0 is returned and errno may be set to [ERANGE].

ERRORS
  The tan() function will fail if:

[ERANGE]        The value to be returned would cause overflow.
  The tan() function may fail if:

EX [EDOM]        The value x is NaN or ±Inf.
EX [ERANGE]      The value to be returned would cause underflow.

EX No other errors will occur.

APPLICATION USAGE
  An application wishing to check for error situations should set errno to 0 before calling tan(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

  The tan() function may lose accuracy when its argument is far from 0.0.

SEE ALSO
  atan(), isnan(), <math.h>.

CHANGE HISTORY
  First released in Issue 1.
  Derived from Issue 1 of the SVID.

Issue 4
  The following changes are incorporated in this issue:

  • Removed references to matherr().

  • The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the ISO C standard and to rationalise error handling in the mathematics functions.

  • The return value specified for [EDOM] is marked as an extension.
NAME
tanh — hyperbolic tangent function

SYNOPSIS
#include <math.h>
double tanh(double x);

DESCRIPTION
The tanh() function computes the hyperbolic tangent of x.

RETURN VALUE
Upon successful completion, tanh() returns the hyperbolic tangent of x.

EX
If x is NaN, NaN is returned and errno may be set to [EDOM].
If the correct value would cause underflow, 0.0 is returned and errno may be set to [ERANGE].

ERRORS
The tanh() function may fail if:

EX [EDOM] The value of x is NaN.
[ERANGE] The correct result would cause underflow.

EX No other errors will occur.

APPLICATION USAGE
An application wishing to check for error situations should set errno to 0 before calling tanh(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

SEE ALSO
atanh(), isnan(), tan(), <math.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

• Removed references to matherr().
• The RETURN VALUE and ERRORS sections are substantially rewritten for alignment with the ISO C standard and to rationalise error handling in the mathematics functions.
• The return value specified for [EDOM] is marked as an extension.
NAME
tcdrain — wait for transmission of output

SYNOPSIS
#include <termios.h>

int tcdrain(int fildes);

DESCRIPTION
The tcdrain() function waits until all output written to the object referred to by fildes is transmitted. The fildes argument is an open file descriptor associated with a terminal.

Any attempts to use tcdrain() from a process which is a member of a background process group on a fildes associated with its controlling terminal, will cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is allowed to perform the operation, and no signal is sent.

RETURN VALUE
Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS
The tcdrain() function will fail if:

[EBADF] The fildes argument is not a valid file descriptor.

[EINTR] A signal interrupted tcdrain().

[ENOTTY] The file associated with fildes is not a terminal.

The tcdrain() function may fail if:

[EIO] The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

FUTURE DIRECTIONS
In the ISO POSIX-1 standard, the possibility of an [EIO] error occurring is described in Section 7.1.1.4, Terminal Access Control, but it is not mentioned in the tcdrain() interface definition. It has become clear that this omission was unintended, so it is likely that the [EIO] error will be reclassified as a “will fail” when the POSIX standard is next updated.

SEE ALSO
tcflush(), <termios.h>, <unistd.h>, the XBD specification, Chapter 9, General Terminal Interface.

CHANGE HISTORY
First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

Issue 4
The following change is incorporated for alignment with the FIPS requirements:

- The words “If _POSIX_JOB_CONTROL is defined” are removed from the start of the second paragraph in the DESCRIPTION section. This is because job control is defined as mandatory for Issue 4 conforming implementations.
Other changes are incorporated as follows:

- The [EIO] error is added to the **ERRORS** section.
- The **FUTURE DIRECTIONS** section is added.
NAME
tcflow — suspend or restart the transmission or reception of data

SYNOPSIS
#include <termios.h>

int tcflow(int fildes, int action);

DESCRIPTION
The tcflow() function suspends transmission or reception of data on the object referred to by
fildes, depending on the value of action. The fildes argument is an open file descriptor associated
with a terminal.

• If action is TCOFF, output is suspended.
• If action is TCOON, suspended output is restarted.
• If action is TCIOFF, the system transmits a STOP character, which is intended to cause the
terminal device to stop transmitting data to the system.
• If action is TCION, the system transmits a START character, which is intended to cause the
terminal device to start transmitting data to the system.

The default on the opening of a terminal file is that neither its input nor its output are
suspended.

Attempts to use tcflow() from a process which is a member of a background process group on a
fildes associated with its controlling terminal, will cause the process group to be sent a SIGTTOU
signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is allowed to
perform the operation, and no signal is sent.

RETURN VALUE
Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to indicate
the error.

ERRORS
The tcflow() function will fail if:

[EBADF] The fildes argument is not a valid file descriptor.
[EINVAL] The action argument is not a supported value.
[ENOTTY] The file associated with fildes is not a terminal.

The tcflow() function may fail if:

[EIO] The process group of the writing process is orphaned, and the writing process
is not ignoring or blocking SIGTTOU.

FUTURE DIRECTIONS
In the ISO POSIX-1 standard, the possibility of an [EIO] error occurring is described in Section
7.1.1.4, Terminal Access Control, but it is not mentioned in the tcflow() interface definition. It has
become clear that this omission was unintended, so it is likely that the [EIO] error will be re-
classified as a “will fail” when the POSIX standard is next updated.

SEE ALSO
tcsendbreak(), <termios.h>, <unistd.h>, the XBD specification, Chapter 9, General Terminal
Interface.
CHANGE HISTORY
First released in Issue 3.
Entry included for alignment with the POSIX.1-1988 standard.

Issue 4
The following change is incorporated for alignment with the FIPS requirements:
- The words “If _POSIX_JOB_CONTROL is defined” are removed from the start of the second paragraph in the DESCRIPTION section. This is because job control is defined as mandatory for Issue 4 conforming implementations.

Other changes are incorporated as follows:
- The descriptions of TCIOFF and TCION are reworded, indicating the intended consequences of transmitting stop and start characters. Issue 3 implied that these consequences were guaranteed.
- The [EIO] error is added to the ERRORS section.
- The FUTURE DIRECTIONS section is added.
NAME
tcflush — flush non-transmitted output data, non-read input data or both

SYNOPSIS
#include <termios.h>

int tcflush(int fildes, int queue_selector);

DESCRIPTION
Upon successful completion, tcflush() discards data written to the object referred to by fildes (an open file descriptor associated with a terminal) but not transmitted, or data received but not read, depending on the value of queue_selector:

- If queue_selector is TCIFLUSH it flushes data received but not read.
- If queue_selector is TCOFLUSH it flushes data written but not transmitted.
- If queue_selector is TCIOFLUSH it flushes both data received but not read and data written but not transmitted.

FIPS
Attempts to use tcflush() from a process which is a member of a background process group on a fildes associated with its controlling terminal, will cause the process group to be sent a SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is allowed to perform the operation, and no signal is sent.

RETURN VALUE
Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS
The tcflush() function will fail if:

- [EBADF] The fildes argument is not a valid file descriptor.
- [EINVAL] The queue_selector argument is not a supported value.
- [ENOTTY] The file associated with fildes is not a terminal.

The tcflow() function may fail if:

- [EIO] The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

FUTURE DIRECTIONS
In the ISO POSIX-1 standard, the possibility of an [EIO] error occurring is described in Section 7.1.1.4, Terminal Access Control, but it is not mentioned in the tcflow() interface definition. It has become clear that this omission was unintended, so it is likely that the [EIO] error will be reclassified as a “will fail” when the POSIX standard is next updated.

SEE ALSO
tcdrain(), <termios.h>, <unistd.h>, the XBD specification, Chapter 9, General Terminal Interface.

CHANGE HISTORY
First released in Issue 3.
Entry included for alignment with the POSIX.1-1988 standard.
Issue 4

The following change is incorporated for alignment with the FIPS requirements:

- The words “If _POSIX_JOB_CONTROL is defined” are removed from the start of the second paragraph in the DESCRIPTION section. This is because job control is defined as mandatory for Issue 4 conforming implementations.

Other changes are incorporated as follows:

- The DESCRIPTION section is modified to indicate that the flush operation will only result if the call to tcflush() is successful.
- The [EIO] error is added to the ERRORS section.
- The FUTURE DIRECTIONS section is added.
NAME
tcgetattr — get the parameters associated with the terminal

SYNOPSIS
#include <termios.h>

int tcgetattr(int fildes, struct termios *termios_p);

DESCRIPTION
The tcgetattr() function gets the parameters associated with the terminal referred to by fildes and stores them in the termios structure referenced by termios_p. The fildes argument is an open file descriptor associated with a terminal.

The termios_p argument is a pointer to a termios structure.

The tcgetattr() operation is allowed from any process.

If the terminal device supports different input and output baud rates, the baud rates stored in the termios structure returned by tcgetattr() reflect the actual baud rates, even if they are equal. If differing baud rates are not supported, the rate returned as the output baud rate is the actual baud rate. If the terminal device does not support split baud rates, the input baud rate stored in the termios structure will be 0.

RETURN VALUE
Upon successful completion, 0 is returned. Otherwise, -1 is returned and errno is set to indicate the error.

ERRORS
The tcgetattr() function will fail if:

[EBADF] The fildes argument is not a valid file descriptor.
[ENOTTY] The file associated with fildes is not a terminal.

FUTURE DIRECTIONS
In a future issue of this document, implementations which do not support differing baud rates will be prohibited from returning 0 as the input baud rate.

SEE ALSO
tcsetattr(), <termios.h>, the XBD specification, Chapter 9, General Terminal Interface.

CHANGE HISTORY
First released in Issue 3.
Entry included for alignment with the POSIX.1-1988 standard.

Issue 4
The following change is incorporated in this issue:

• The FUTURE DIRECTIONS section is added to allow for alignment with the ISO POSIX-1 standard.
NAME
tcgetpgrp — get foreground process group ID

SYNOPSIS
#include <sys/types.h>
#include <unistd.h>

pid_t tcgetpgrp(int fildes);

DESCRIPTION
The tcgetpgrp() function will return the value of the process group ID of the foreground process

group associated with the terminal.

If there is no foreground process group, tcgetpgrp() returns a value greater than 1 that does not
match the process group ID of any existing process group.

The tcgetpgrp() function is allowed from a process that is a member of a background process
group; however, the information may be subsequently changed by a process that is a member of
a foreground process group.

RETURN VALUE
Upon successful completion, tcgetpgrp() returns the value of the process group ID of the
foreground process associated with the terminal. Otherwise, −1 is returned and errno is set to
indicate the error.

ERRORS
The tcgetpgrp() function will fail if:

[EBADF] The fildes argument is not a valid file descriptor.
[ENOTTY] The calling process does not have a controlling terminal, or the file is not the
controlling terminal.

SEE ALSO
setsid(), setpgid(), tcsetpgrp(), <sys/types.h>, <unistd.h>.

CHANGE HISTORY
First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

Issue 4
The following change is incorporated for alignment with the FIPS requirements:

• The DESCRIPTION section is clarified and the phrase “If _POSIX_JOB_CONTROL is
defined” is removed because job control is now mandatory on all XSI-conformant systems.

Other changes are incorporated as follows:

• The header <sys/types.h> is now marked as optional (OH); this header need not be included
on XSI-conformant systems.

• The header <unistd.h> is added to the SYNOPSIS section.
NAME
tcgetsid — get process group ID for session leader for controlling terminal

SYNOPSIS
UX

```c
#include <termios.h>

pid_t tcgetsid(int fildes);
```

DESCRIPTION
The tcgetsid() function obtains the process group ID of the session for which the terminal specified by fildes is the controlling terminal.

RETURN VALUE
Upon successful completion, tcgetsid() returns the process group ID associated with the terminal. Otherwise, a value of (pid_t)-1 is returned and errno is set to indicate the error.

ERRORS
The tcgetsid() function will fail if:

- [EACCESS] The fildes argument is not associated with a controlling terminal.
- [EBADF] The fildes argument is not a valid file descriptor.
- [ENOTTY] The file associated with fildes is not a terminal.

SEE ALSO
<termios.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
tcsendbreak — send a “break” for a specific duration

SYNOPSIS
#include <termios.h>

int tcsendbreak(int fildes, int duration);

DESCRIPTION
The fildes argument is an open file descriptor associated with a terminal.
If the terminal is using asynchronous serial data transmission, tcsendbreak() will cause
transmission of a continuous stream of zero-valued bits for a specific duration. If duration is 0, it
will cause transmission of zero-valued bits for at least 0.25 seconds, and not more than 0.5
seconds. If duration is not 0, it will send zero-valued bits for an implementation-dependent
period of time.
If the terminal is not using asynchronous serial data transmission, it is implementation-
dependent whether tcsendbreak() sends data to generate a break condition or returns without
taking any action.

FIPS
Attempts to use tcsendbreak() from a process which is a member of a background process group
on a fildes associated with its controlling terminal, will cause the process group to be sent a
SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is
allowed to perform the operation, and no signal is sent.

RETURN VALUE
Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to indicate
the error.

ERRORS
The tcsendbreak() function will fail if:
[EBADF] The fildes argument is not a valid file descriptor.
[ENOTTY] The file associated with fildes is not a terminal.
The tcsendbreak() function may fail if:
[EIO] The process group of the writing process is orphaned, and the writing process
is not ignoring or blocking SIGTTOU.

FUTURE DIRECTIONS
In the ISO POSIX-1 standard, the possibility of an [EIO] error occurring is described in Section
7.1.1.4, Terminal Access Control, but it is not mentioned in the tcsendbreak() interface definition.
It has become clear that this omission was unintended, so it is likely that the [EIO] error will be
reclassified as a “will fail” when the POSIX standard is next updated.

SEE ALSO
<termios.h>, <unistd.h>, the XBD specification, Chapter 9, General Terminal Interface.

CHANGE HISTORY
First released in Issue 3.
Entry included for alignment with the POSIX.1-1988 standard.

Issue 4
The following change is incorporated for alignment with the FIPS requirements:
• In the DESCRIPTION section the phrase “If _POSIX_JOB_CONTROL is defined” is
removed because job control is now mandatory on all XSI-conformant systems.
Another change is incorporated as follows:

- The [EIO] error is added to the **ERRORS** section.
NAME
tcsetattr — set the parameters associated with the terminal

SYNOPSIS
#include <termios.h>

int tcsetattr(int fildes, int optional_actions,
               const struct termios *termios_p);

DESCRIPTION
The tcsetattr() function sets the parameters associated with the terminal referred to by the open
file descriptor fildes (an open file descriptor associated with a terminal) from the termios
structure referenced by termios_p as follows:

• If optional_actions is TCSANOW, the change will occur immediately.

• If optional_actions is TCSADRAIN, the change will occur after all output written to fildes is
  transmitted. This function should be used when changing parameters that affect output.

• If optional_actions is TCSAFLUSH, the change will occur after all output written to fildes is
  transmitted, and all input so far received but not read will be discarded before the change is
  made.

If the output baud rate stored in the termios structure pointed to by termios_p is the zero baud
rate, B0, the modem control lines will no longer be asserted. Normally, this will disconnect the
line.

If the input baud rate stored in the termios structure pointed to by termios_p is 0, the input baud
rate given to the hardware will be the same as the output baud rate stored in the termios
structure.

The tcsetattr() function will return successfully if it was able to perform any of the requested
actions, even if some of the requested actions could not be performed. It will set all the attributes
that implementation supports as requested and leave all the attributes not supported by the
implementation unchanged. If no part of the request can be honoured, it will return −1 and set
errno to [EINVAL]. If the input and output baud rates differ and are a combination that is not
supported, neither baud rate is changed. A subsequent call to tcgetattr() will return the actual
state of the terminal device (reflecting both the changes made and not made in the previous
tcsetattr() call). The tcsetattr() function will not change the values in the termios structure
whether or not it actually accepts them.

The effect of tcsetattr() is undefined if the value of the termios structure pointed to by termios_p
was not derived from the result of a call to tcgetattr() on fildes; an application should modify
only fields and flags defined by this document between the call to tcgetattr() and tcsetattr(),
leaving all other fields and flags unmodified.

No actions defined by this document, other than a call to tcsetattr() or a close of the last file
descriptor in the system associated with this terminal device, will cause any of the terminal
attributes defined by this document to change.

FIPS
Attempts to use tcsetattr() from a process which is a member of a background process group on
a fildes associated with its controlling terminal, will cause the process group to be sent a
SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is
allowed to perform the operation, and no signal is sent.

RETURN VALUE
Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to indicate
the error.
The `tcsetattr()` function will fail if:

- **[EBADF]** The `fildes` argument is not a valid file descriptor.
- **[EINTR]** A signal interrupted `tcsetattr()`.
- **[EINVAL]** The `optional_actions` argument is not a supported value, or an attempt was made to change an attribute represented in the `termios` structure to an unsupported value.
- **[ENOTTY]** The file associated with `fildes` is not a terminal.

The `tcsetattr()` function may fail if:

- **[EIO]** The process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU.

**APPLICATION USAGE**

If trying to change baud rates, applications should call `tcsetattr()` then call `tcgetattr()` in order to determine what baud rates were actually selected.

**FUTURE DIRECTIONS**

Using an input baud rate of 0 to set the input rate equal to the output rate may not be supported in a future issue of this document.

In the ISO POSIX-1 standard, the possibility of an [EIO] error occurring is described in Section 7.1.1.4, Terminal Access Control, but it is not mentioned in the `tcsetattr()` interface definition. It has become clear that this omission was unintended, so it is likely that the [EIO] error will be reclassified as a “will fail” when the POSIX standard is next updated.

**SEE ALSO**

`cfgetispeed()`, `tcgetattr()`, `<termios.h>`, `<unistd.h>`, the XBD specification, Chapter 9, General Terminal Interface.

**CHANGE HISTORY**

First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

**Issue 4**

The following change is incorporated for alignment with the ISO POSIX-1 standard:

- The argument `termios_p` is changed from type `struct termios *` to `const struct termios *`.

The following change is incorporated for alignment with the FIPS requirements:

- In the DESCRIPTION section the phrase “If `_POSIX_JOB_CONTROL` is defined” is removed because job control is now mandatory on all XSI-conformant systems.

Other changes are incorporated as follows:

- The words “and stores them in” are changed to “from” in the first paragraph of the DESCRIPTION section.
- The [EINTR] and [EIO] errors are added to the ERRORS section.
- The FUTURE DIRECTIONS section is added to allow for alignment with the ISO POSIX-1 standard.
NAME
tcsetpgrp — set foreground process group ID

SYNOPSIS
OH
#include <sys/types.h>
#include <unistd.h>

int tcsetpgrp(int fildes, pid_t pgid_id);

DESCRIPTION
FIPS If the process has a controlling terminal, tcsetpgrp() will set the foreground process group ID
associated with the terminal to pgid_id. The file associated with fildes must be the controlling
terminal of the calling process and the controlling terminal must be currently associated with the
session of the calling process. The value of pgid_id must match a process group ID of a process
in the same session as the calling process.

RETURN VALUE
Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to indicate
the error.

ERRORS
The tcsetpgrp() function will fail if:
[EBADF] The fildes argument is not a valid file descriptor.
[EINVAL] This implementation does not support the value in the pgid_id argument.
[ENOTTY] The calling process does not have a controlling terminal, or the file is not the
controlling terminal, or the controlling terminal is no longer associated with the
session of the calling process.

FIPS [EPERM] The value of pgid_id does not match the process group ID of a process in the
same session as the calling process.

SEE ALSO
tcgetpgrp(), <sys/types.h>, <unistd.h>.

CHANGE HISTORY
First released in Issue 3.
Entry included for alignment with the POSIX.1-1988 standard.

Issue 4
The following change is incorporated for alignment with the FIPS requirements:
• In the DESCRIPTION section the phrase “If _POSIX_JOB_CONTROL is defined” is
removed because job control is now mandatory on all XSI-conformant systems.

Other changes are incorporated as follows:
• The header <sys/types.h> is now marked as optional (OH); this header need not be included
on XSI-conformant systems.
• The header <unistd.h> is added to the SYNOPSIS section.
• The [ENOSYS] error is removed from the ERRORS section.
NAME
tdelete — delete node from binary search tree

SYNOPSIS

```c
#include <search.h>

void *tdelete(const void *key, void **rootp,
              int (*compar)(const void *, const void *));
```

DESCRIPTION

Refer to tsearch().

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following change is incorporated in this issue:

- The function return value is changed from char * to void*, the type of argument key is changed from char * to const void*, rootp is changed from char ** to void**, and arguments to compar() are formally defined.
NAME
telldir — current location of a named directory stream

SYNOPSIS
EX
#include <dirent.h>
long int telldir(DIR *dirp);

DESCRIPTION
The telldir() function obtains the current location associated with the directory stream specified by dirp.

UX If the most recent operation on the directory stream was a seekdir(), the directory position returned from the telldir() is the same as that supplied as a loc argument for seekdir().

RETURN VALUE
Upon successful completion, telldir() returns the current location of the specified directory stream.

ERRORS
No errors are defined.

SEE ALSO
opendir(), readdir(), seekdir(), <dirent.h>.

CHANGE HISTORY
First released in Issue 2.

Issue 4
The following changes are incorporated in this issue:

- The header <sys/types.h> is removed from the SYNOPSIS section.
- The function return value is expanded to long int.

Issue 4, Version 2
The DESCRIPTION is updated for X/OPEN UNIX conformance to indicate that a call to telldir() immediately following a call to seekdir(), returns the loc value passed to the seekdir() call.
tempnam() — create a name for a temporary file

SYNOPSIS

```
#include <stdio.h>

char *tempnam(const char *dir, const char *pfx);
```

DESCRIPTION

The `tempnam()` function generates a pathname that may be used for a temporary file.

The `tempnam()` function allows the user to control the choice of a directory. The `dir` argument points to the name of the directory in which the file is to be created. If `dir` is a null pointer or points to a string which is not a name for an appropriate directory, the path prefix defined as `{P_tmpdir}` in the `<stdio.h>` header is used. If that directory is not accessible, an implementation-dependent directory may be used.

Many applications prefer their temporary files to have certain initial letter sequences in their names. The `pfx` argument should be used for this. This argument may be a null pointer or point to a string of up to five bytes to be used as the beginning of the filename.

RETURN VALUE

Upon successful completion, `tempnam()` allocates space for a string, puts the generated pathname in that space and returns a pointer to it. The pointer is suitable for use in a subsequent call to `free()` . Otherwise it returns a null pointer and sets `errno` to indicate the error.

ERRORS

The `tempnam()` function will fail if:

- `[ENOMEM]` Insufficient storage space is available.

APPLICATION USAGE

This function only creates pathnames. It is the application’s responsibility to create and remove the files. Between the time a pathname is created and the file is opened, it is possible for some other process to create a file with the same name. Applications may find `tmpfile()` more useful.

Some implementations of `tempnam()` may use `tmpnam()` internally. On such implementations, if called more than `{TMP_MAX}` times in a single process, the behaviour is implementation-dependent.

SEE ALSO

`fopen()`, `free()`, `open()`, `tmpfile()`, `tmpnam()`, `unlink()`, `<stdio.h>`.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated in this issue:

- The type of arguments `dir` and `pfx` is changed from `char *` to `const char *`.
- The `DESCRIPTION` section is changed to indicate that `pfx` is treated as a string of bytes and not as a string of (possibly multi-byte) characters.
- The second paragraph of the `APPLICATION USAGE` section is expanded.
NAME
tfind — search binary search tree

SYNOPSIS
EX
#include <search.h>

void *tfind(const void * key, void *const * rootp,
            int (*compar)(const void *, const void *));

DESCRIPTION
Refer to tsearch().

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:
- The function return value is changed from char * to void*.
- The type of argument key is changed from char * to const void*; the type of argument rootp is changed from char ** to void* const*.
- Arguments to compar() are formally defined.
NAME
time — get time

SYNOPSIS
#include <time.h>

time_t time(time_t *tloc);

DESCRIPTION
The time() function returns the value of time in seconds since the Epoch.
The tloc argument points to an area where the return value is also stored. If tloc is a null pointer, no value is stored.

RETURN VALUE
Upon successful completion, time() returns the value of time. Otherwise, (time_t)–1 is returned.

ERRORS
No errors are defined.

SEE ALSO
asctime(), clock(), ctime(), difftime(), gmtime(), localtime(), mktime(), strftime(), strptime(), utime(), <time.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:
• The RETURN VALUE section is updated to indicate that (time_t)–1 will be returned on error.
NAME
times — get process and waited-for child process times

SYNOPSIS
#include <sys/times.h>
clock_t times(struct tms *buffer);

DESCRIPTION
The times() function fills the tms structure pointed to by buffer with time-accounting
information. The structure tms is defined in <sys/times.h>.

All times are measured in terms of the number of clock ticks used.

The times of a terminated child process are included in the tms_cutime and tms_cstime
elements of the parent when wait() or waitpid() returns the process ID of this terminated child.
If a child process has not waited for its children, their times will not be included in its times.

• The tms_utime structure member is the CPU time charged for the execution of user
instructions of the calling process.

• The tms_stime structure member is the CPU time charged for execution by the system on
behalf of the calling process.

• The tms_cutime structure member is the sum of the tms_utime and tms_cutime times of the
child processes.

• The tms_cstime structure member is the sum of the tms_stime and tms_cstime times of the
child processes.

RETURN VALUE
Upon successful completion, times() returns the elapsed real time, in clock ticks, since an
arbitrary point in the past (for example, system start-up time). This point does not change from
one invocation of times() within the process to another. The return value may overflow the
possible range of type clock_t. If times() fails, (clock_t)−1 is returned and errno is set to indicate
the error.

ERRORS
No errors are defined.

APPLICATION USAGE
Applications should use sysconf(_SC_CLK_TCK) to determine the number of clock ticks per
second as it may vary from system to system.

SEE ALSO
exec, fork(), sysconf(), time(), wait(), <sys/times.h>.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated for alignment with the ISO POSIX-1 standard:

• All references to the constant [CLK_TCK] are removed.

• The RETURN VALUE section is updated to indicate that (clock_t)−1 will be returned on
error.
NAME
 timezone — difference from UTC and local standard time

SYNOPSIS
 EX

```c
#include <time.h>

extern long int timezone;
```

DESCRIPTION
 Refer to `tzset()`.

CHANGE HISTORY
 First released in Issue 1.
 Derived from Issue 1 of the SVID.

Issue 4
 The following changes are incorporated in this issue:

- In the NAME section, “GMT” is changed to “UTC”.
- The interface is marked as an extension.
- The type of `timezone` is expanded to `extern long int`.

NAME
tmpfile — create a temporary file

SYNOPSIS
#include <stdio.h>
FILE *tmpfile(void);

DESCRIPTION
The tmpfile() function creates a temporary file and opens a corresponding stream. The file will
automatically be deleted when all references to the file are closed. The file is opened as in
fopen() for update (w+).

RETURN VALUE
Upon successful completion, tmpfile() returns a pointer to the stream of the file that is created.
Otherwise, it returns a null pointer and sets errno to indicate the error.

ERRORS
The tmpfile() function will fail if:

[EINTR] A signal was caught during tmpfile().
[EMFILE] OPEN_MAX file descriptors are currently open in the calling process.
[ENFILE] The maximum allowable number of files is currently open in the system.
[ENOSPC] The directory or file system which would contain the new file cannot be expanded.

The tmpfile() function may fail if:

EX [EMFILE] FOPEN_MAX streams are currently open in the calling process.
[ENOMEM] Insufficient storage space is available.

APPLICATION USAGE
The stream refers to a file which is unlinked. If the process is killed in the period between file
creation and unlinking, a permanent file may be left behind.

On some implementations, an error message may be printed if the stream cannot be opened.

SEE ALSO
fopen(), tmpnam(), unlink(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

• The argument list is explicitly defined as void.
• The [EINTR] error is moved to the “will fail” part of the ERRORS section; [EMFILE],
[ENFILE] and [ENOSPC] are no longer marked as extensions; [EACCES], [ENOTDIR] and
[EROFS] are removed; and the [EMFILE] error in the “may fail” part is marked as an
extension.
tmpnam() — create a name for a temporary file

#include <stdio.h>
char *tmpnam(char * s);

The tmpnam() function generates a string that is a valid filename and that is not the same as the name of an existing file.

The tmpnam() function generates a different string each time it is called from the same process, up to {TMP_MAX} times. If it is called more than {TMP_MAX} times, the behaviour is implementation-dependent.

The implementation will behave as if no function defined in this document calls tmpnam().

Upon successful completion, tmpnam() returns a pointer to a string.

If the argument s is a null pointer, tmpnam() leaves its result in an internal static object and returns a pointer to that object. Subsequent calls to tmpnam() may modify the same object. If the argument s is not a null pointer, it is presumed to point to an array of at least {L_tmpnam} chars; tmpnam() writes its result in that array and returns the argument as its value.

No errors are defined.

This function only creates filenames. It is the application’s responsibility to create and remove the files.

Between the time a pathname is created and the file is opened, it is possible for some other process to create a file with the same name. Applications may find tmpfile() more useful.

SEE ALSO
fopen(), open(), tempnam(), tmpfile(), unlink(), <stdio.h>.

First released in Issue 1.
Derived from Issue 1 of the SVID.
NAME
toascii — translate integer to a 7-bit ASCII character

SYNOPSIS
EX
#include <ctype.h>
int toascii(int c);

DESCRIPTION
The toascii() function converts its argument into a 7-bit ASCII character.

RETURN VALUE
The toascii() function returns the value (c & 0x7f).

ERRORS
No errors are returned.

SEE ALSO
isascii(), <ctype.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.
NAME
_tolower — transliterate upper-case characters to lower-case

SYNOPSIS

```c
#include <ctype.h>

int _tolower(int c);
```

DESCRIPTION

The _tolower() macro is equivalent to tolower(c) except that the argument c must be an upper-case letter.

RETURN VALUE

On successful completion, _tolower() returns the lower-case letter corresponding to the argument passed.

ERRORS

No errors are defined.

SEE ALSO

tolower(), isupper(), <ctype.h>, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following change is incorporated in this issue:

- The RETURN VALUE section is expanded.
NAME
tolower — transliterate upper-case characters to lower-case

SYNOPSIS
#include <ctype.h>
int tolower(int c);

DESCRIPTION
The tolower() function has as a domain a type int, the value of which is
representable as an unsigned char or the value of EOF. If the argument has any
other value, the behaviour is undefined. If the argument of tolower() represents
an upper-case letter, and there exists a corresponding lower-case letter (as
defined by character type information in the program locale category LC_CTYPE),
the result is the corresponding lower-case letter. All other arguments in
the domain are returned unchanged.

RETURN VALUE
On successful completion, tolower() returns the lower-case letter corresponding
to the argument passed; otherwise it returns the argument unchanged.

ERRORS
No errors are defined.

SEE ALSO
setlocale(), <ctype.h>, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:
• Reference to “shift information” is replaced by “character type information”.
• The RETURN VALUE section is added.
NAME
_toupper — transliterate lower-case characters to upper-case

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

DESCRIPTION
The _toupper() macro is equivalent to toupper() except that the argument c must be a lower-case letter.

RETURN VALUE
On successful completion, _toupper() returns the upper-case letter corresponding to the argument passed.

ERRORS
No errors are defined.

SEE ALSO
islower(), toupper(), <ctype.h>, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:
- The RETURN VALUE section is expanded.
NAME
toupper — transliterate lower-case characters to upper-case

SYNOPSIS
#include <ctype.h>
int toupper(int c);

DESCRIPTION
The toupper() function has as a domain a type int, the value of which is representable as an unsigned char or the value of EOF. If the argument has any other value, the behaviour is undefined. If the argument of toupper() represents a lower-case letter, and there exists a corresponding upper-case letter (as defined by character type information in the program locale category LC_CTYPE), the result is the corresponding upper-case letter. All other arguments in the domain are returned unchanged.

RETURN VALUE
On successful completion, toupper() returns the upper-case letter corresponding to the argument passed.

ERRORS
No errors are defined.

SEE ALSO
setlocale(), <ctype.h>, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:
• Reference to “shift information” is replaced by “character type information”.
• The RETURN VALUE section is added.
towlower()

NAME
towlower — transliterate upper-case wide-character code to lower-case

SYNOPSIS
#include <wchar.h>

wint_t towlower(wint_t wc);

DESCRIPTION
The towlower() function has as a domain a type wint_t, the value of which must be a character representable as a wchar_t, and must be a wide-character code corresponding to a valid character in the current locale or the value of WEOF. If the argument has any other value, the behaviour is undefined. If the argument of towlower() represents an upper-case wide-character code, and there exists a corresponding lower-case wide-character code (as defined by character type information in the program locale category LC_CTYPE), the result is the corresponding lower-case wide-character code. All other arguments in the domain are returned unchanged.

RETURN VALUE
On successful completion, towlower() returns the lower-case letter corresponding to the argument passed; otherwise it returns the argument unchanged.

ERRORS
No errors are defined.

SEE ALSO
setlocale(), <wchar.h>, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
towupper — transliterate lower-case wide-character code to upper-case

SYNOPSIS

```c
#include <wchar.h>

wint_t towupper(wint_t wc);
```

DESCRIPTION

The `towupper()` function has as a domain a type `wint_t`, the value of which must be a character representable as a `wchar_t`, and must be a wide-character code corresponding to a valid character in the current locale or the value of WEOF. If the argument has any other value, the behaviour is undefined. If the argument of `towupper()` represents a lower-case wide-character code, and there exists a corresponding upper-case wide-character code (as defined by character type information in the program locale category LC_CTYPE), the result is the corresponding upper-case wide-character code. All other arguments in the domain are returned unchanged.

RETURN VALUE

Upon successful completion, `towupper()` returns the upper-case letter corresponding to the argument passed. Otherwise it returns the argument unchanged.

ERRORS

No errors are defined.

SEE ALSO

`setlocale()`, `<wchar.h>`, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY

First released in Issue 4.

Derived from the MSE working draft.
NAME
truncate — truncate a file to a specified length

SYNOPSIS
UX

```
#include <unistd.h>

int truncate(const char *path, off_t length);
```

DESCRIPTION
Refer to ftruncate().

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
tdelete, tfind, tsearch, twalk — manage binary search tree

SYNOPSIS
EX
```
#include <search.h>

void *tsearch(const void * key, void ** rootp,
               int (*compar)(const void *, const void *));

void *tfind(const void * key, void *const * rootp,
            int(* compar)(const void *, const void *));

void *tdelete(const void * key, void ** rootp,
               int(*compar)(const void *, const void *));

void twalk(const void * root,
           void (*action)(const void *, VISIT, int));
```

DESCRIPTION

The **tsearch()**, **tfind()**, **tdelete()** and **twalk()** functions manipulate binary search trees. Comparisons are made with a user-supplied routine, the address of which is passed as the **compar** argument. This routine is called with two arguments, the pointers to the elements being compared. The user-supplied routine must return an integer less than, equal to or greater than 0, according to whether the first argument is to be considered less than, equal to or greater than the second argument. The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

The **tsearch()** function is used to build and access the tree. The **key** argument is a pointer to an element to be accessed or stored. If there is a node in the tree whose element is equal to the value pointed to by **key**, a pointer to this found node is returned. Otherwise, the value pointed to by **key** is inserted (that is, a new node is created and the value of **key** is copied to this node), and a pointer to this node returned. Only pointers are copied, so the calling routine must store the data. The **rootp** argument points to a variable that points to the root node of the tree. A null pointer value for the variable pointed to by **rootp** denotes an empty tree; in this case, the variable will be set to point to the node which will be at the root of the new tree.

Like **tsearch()**, **tfind()** will search for a node in the tree, returning a pointer to it if found. However, if it is not found, **tfind()** will return a null pointer. The arguments for **tfind()** are the same as for **tsearch()**.

The **tdelete()** function deletes a node from a binary search tree. The arguments are the same as for **tsearch()**. The variable pointed to by **rootp** will be changed if the deleted node was the root of the tree. The **tdelete()** function returns a pointer to the parent of the deleted node, or a null pointer if the node is not found.

The **twalk()** function traverses a binary search tree. The **root** argument is a pointer to the root node of the tree to be traversed. (Any node in a tree may be used as the root for a walk below that node.) The argument **action** is the name of a routine to be invoked at each node. This routine is, in turn, called with three arguments. The first argument is the address of the node being visited. The structure pointed to by this argument is unspecified and must not be modified by the application, but it is guaranteed that a pointer-to-node can be converted to pointer-to-pointer-to-element to access the element stored in the node. The second argument is a value from an enumeration data type:

```
typedef enum { preorder, postorder, endorder, leaf } VISIT;
```

(defined in `<search.h>`), depending on whether this is the first, second or third time that the node is visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a
leaf. The third argument is the level of the node in the tree, with the root being level 0.

RETURN VALUE
If the node is found, both tsearch() and tfind() return a pointer to it. If not, tfind() returns a null pointer, and tsearch() returns a pointer to the inserted item.

A null pointer is returned by tsearch() if there is not enough space available to create a new node.

A null pointer is returned by tsearch(), tfind() and tdelete() if rootp is a null pointer on entry.

The tdelete() function returns a pointer to the parent of the deleted node, or a null pointer if the node is not found.

The twalk() function returns no value.

ERRORS
No errors are defined.

EXAMPLES
The following code reads in strings and stores structures containing a pointer to each string and a count of its length. It then walks the tree, printing out the stored strings and their lengths in alphabetical order.

```
#include <search.h>
#include <string.h>
#include <stdio.h>
#define STRSZ 10000
#define NODSZ 500

struct node { /* pointers to these are stored in the tree */
    char *string;
    int   length;
};

char  string_space[STRSZ]; /* space to store strings */
struct node nodes[NODSZ];  /* nodes to store */
void *root = NULL;        /* this points to the root */

int main(int argc, char *argv[])
{
    char    *strptr = string_space;
    struct node *nodeptr = nodes;
    void     print_node(const void *, VISIT, int);
    int     i = 0, node_compare(const void *, const void *, const void *);

    while (gets(strptr) != NULL && i++ < NODSZ) {
        /* set node */
        nodeptr->string = strptr;
        nodeptr->length = strlen(strptr);
        /* put node into the tree */
        (void) tsearch((void *)nodeptr, (void **)&root,
                      node_compare);
        /* adjust pointers, so we do not overwrite tree */
        strptr += nodeptr->length + 1;
        nodeptr++;
    }

    twalk(root, print_node);
    return 0;
}
```

674 X/Open CAE Specification (1994)
/*
 * This routine compares two nodes, based on an alphabetical ordering of the string field.
 */
int
node_compare(const void *node1, const void *node2)
{
    return strcmp(((const struct node *) node1)->string,
                   ((const struct node *) node2)->string);
}

/*
 * This routine prints out a node, the second time twalk encounters it or if it is a leaf.
 */
void
print_node(const void *ptr, VISIT order, int level)
{
    const struct node *p = *(const struct node **) ptr;
    if (order == postorder || order == leaf) {
        (void) printf("string = %s, length = %d\n", p->string, p->length);
    }
}

APPLICATION USAGE
The root argument to twalk() is one level of indirection less than the rootp arguments to tsearch() and tdelete().

There are two nomenclatures used to refer to the order in which tree nodes are visited. The tsearch() function uses preorder, postorder and endorder to refer respectively to visiting a node before any of its children, after its left child and before its right, and after both its children. The alternative nomenclature uses preorder, inorder and postorder to refer to the same visits, which could result in some confusion over the meaning of postorder.

If the calling function alters the pointer to the root, the result is undefined.

SEE ALSO
bsearch(), hsearch(), lsearch(), <search.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:
- The type of argument key in the definition of tsearch() is changed from void* to const void*.
  The definitions of other functions are changed as indicated on their respective entries.
- Various minor wording changes are made in the DESCRIPTION section to improve clarity and accuracy. In particular, additional notes are added about constraints on the first argument to twalk().
• The sample code in the EXAMPLES section is updated to use ISO C syntax. Also the definition of the root and argv items is changed.
• The paragraph in the APPLICATION USAGE section about casts is removed.
NAME
ttyname — find pathname of a terminal

SYNOPSIS
#include <unistd.h>
char *ttyname(int fildes);

DESCRIPTION
The ttyname() function returns a pointer to a string containing a null-terminated pathname of
the terminal associated with file descriptor fildes. The return value may point to static data
whose content is overwritten by each call.

RETURN VALUE
Upon successful completion, ttyname() returns a pointer to a string. Otherwise, a null pointer is
returned and errno is set to indicate the error.

ERRORS
The ttyname() function may fail if:

- [EBADF] The fildes argument is not a valid file descriptor.
- [ENOTTY] The fildes argument does not refer to a terminal device.

SEE ALSO
<unistd.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

- The header <unistd.h> is added to the SYNOPSIS.
- The statement indicating that errno will be set on error in the RETURN VALUE section, and
  the errors [EBADF] and [ENOTTY], are marked as extensions.
NAME
ttyslot — find the slot of the current user in the user accounting database

SYNOPSIS
UX

#include <stdlib.h>

int ttyslot(void); (TO BE WITHDRAWN)

DESCRIPTION
The ttyslot() function returns the index of the current user’s entry in the user accounting
database. The current user’s entry is an entry for which the utline member matches the name of
a terminal device associated with any of the process’ file descriptors 0, 1 or 2. The index is an
ordinal number representing the record number in the database of the current user’s entry. The
first entry in the database is represented by the return value 0.

RETURN VALUE
Upon successful completion, ttyslot() returns the index of the current user’s entry in the user
accounting database. The ttyslot() function returns −1 if an error was encountered while
searching the database or if none of file descriptors 0, 1, or 2 is associated with a terminal device.

ERRORS
No errors are defined.

SEE ALSO
endutxent(), ttyname(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
twalk — traverse binary search tree

SYNOPSIS
#include <search.h>

void twalk(const void * root ,
        void (* action)(const void *, VISIT, int ));

DESCRIPTION
Refer to tsearch().

CHANGE HISTORY
First released in Issue 3.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:
• The type of argument root is changed from char * to const void*, and the argument list to
action() is formally defined.
NAME
tzname — timezone strings

SYNOPSIS
#include <time.h>
extern char *tzname[];

DESCRIPTION
Refer to tzset().

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:
• The header <time.h> is added to the SYNOPSIS section.
NAME  
tzset — set time zone conversion information

SYNOPSIS  
#include <time.h>
void tzset (void);
extern char *tzname[];

extern long int timezone;
extern int daylight;

DESCRIPTION  
The tzset() function uses the value of the environment variable TZ to set time conversion
information used by localtime(), ctime(), strftime() and mktime(). If TZ is absent from the
environment, implementation-dependent default time zone information is used.

The tzset() function sets the external variable tzname as follows:

    tzname[0] = "std";
    tzname[1] = "dst";

where std and dst are as described in the XBD specification, Chapter 6, Environment Variables.

The tzset() function also sets the external variable daylight to 0 if Daylight Savings Time
conversions should never be applied for the time zone in use; otherwise non-zero. The external
variable timezone is set to the difference, in seconds, between Coordinated Universal Time (UTC)
and local standard time, for example:

<table>
<thead>
<tr>
<th>TZ</th>
<th>timezone</th>
</tr>
</thead>
<tbody>
<tr>
<td>EST</td>
<td>5<em>60</em>60</td>
</tr>
<tr>
<td>GMT</td>
<td>0<em>60</em>60</td>
</tr>
<tr>
<td>JST</td>
<td>-9<em>60</em>60</td>
</tr>
<tr>
<td>MET</td>
<td>-1<em>60</em>60</td>
</tr>
<tr>
<td>MST</td>
<td>7<em>60</em>60</td>
</tr>
<tr>
<td>PST</td>
<td>8<em>60</em>60</td>
</tr>
</tbody>
</table>

RETURN VALUE  
The tzset() function returns no value.

ERRORS  
No errors are defined.

SEE ALSO  
ctime(), localtime(), mktime(), strftime(), <time.h>.

CHANGE HISTORY  
First released in Issue 1.
Derived from Issue 1 of the SVID.
Issue 4

The following change is incorporated for alignment with the ISO POSIX-1 standard:

- The argument list is explicitly defined as `void`.

Another change is incorporated as follows:

- The reference to `timezone` in the SYNOPSIS section is marked as an extension.
NAME
ualarm — set the interval timer

SYNOPSIS
UX
#include <unistd.h>

useconds_t ualarm(useconds_t useconds, useconds_t interval);

DESCRIPTION
The ualarm() function causes the SIGALRM signal to be generated for the calling process after the number of real-time microseconds specified by the useconds argument has elapsed. When the interval argument is non-zero, repeated timeout notification occurs with a period in microseconds specified by the interval argument. If the notification signal, SIGALRM, is not caught or ignored, the calling process is terminated.

Implementations may place limitations on the granularity of timer values. For each interval timer, if the requested timer value requires a finer granularity than the implementation supports, the actual timer value will be rounded up to the next supported value.

Interactions between ualarm() and either alarm() or sleep() are unspecified.

RETURN VALUE
The ualarm() function returns the number of microseconds remaining from the previous ualarm() call. If no timeouts are pending or if ualarm() has not previously been called, ualarm() returns 0.

ERRORS
No errors are defined.

APPLICATION USAGE
The ualarm() function is a simplified interface to setitimer(), and uses the ITIMER_REAL interval timer.

SEE ALSO
alarm(), setitimer(), sleep(), <unistd.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
ulimit — get and set process limits

SYNOPSIS
#include <ulimit.h>

long int ulimit(int cmd, ...);

DESCRIPTION
The ulimit() function provides for control over process limits. The cmd values, defined in <ulimit.h> include:

UL_GETFSIZE Return the soft file size limit of the process. The limit is in units of 512-byte blocks and is inherited by child processes. Files of any size can be read. The return value is the integer part of the soft file size limit divided by 512. If the result cannot be represented as a long int, the result is unspecified.

UL_SETFSIZE Set the hard and soft file size limits for output operations of the process to the value of the second argument, taken as a long int. Any process may decrease its own hard limit, but only a process with appropriate privileges may increase the limit. The new file size limit is returned. The hard and soft file size limits are set to the specified value multiplied by 512. If the result would overflow an rlimit_t, the actual value set is unspecified.

RETURN VALUE
Upon successful completion, ulimit() returns the value of the requested limit. Otherwise −1 is returned and errno is set to indicate the error.

ERRORS
The ulimit() function will fail and the limit will be unchanged if:

[EINVAL] The cmd argument is not valid.

[EPERM] A process not having appropriate privileges attempts to increase its file size limit.

APPLICATION USAGE
As all return values are permissible in a successful situation, an application wishing to check for error situations should set errno to 0, then call ulimit(), and, if it returns −1, check to see if errno is non-zero.

SEE ALSO
getrlimit(), setrlimit(), write(), <ulimit.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated in this issue:

• The use of long is replaced by long int in the SYNOPSIS and the DESCRIPTION sections.

Issue 4, Version 2
In the DESCRIPTION, the discussion of UL_GETFSIZE and UL_SETFSIZE is revised generally to distinguish between the soft and the hard file size limit of the process. For UL_GETFSIZE, the return value is defined more precisely. For UL_SETFSIZE, the effect on both file size limits is specified, as is the effect if the result would overflow an rlimit_t.
NAME
umask — set and get file mode creation mask

SYNOPSIS
#include <sys/types.h>
#include <sys/stat.h>
mode_t umask(mode_t cmask);

DESCRIPTION
The umask() function sets the process’ file mode creation mask to cmask and returns the previous
value of the mask. Only the file permission bits of cmask (see <sys/stat.h>) are used; the meaning
of the other bits is implementation-dependent.

The process’ file mode creation mask is used during open(), creat(), mkdir() and mkfifo() to turn
off permission bits in the mode argument supplied. Bit positions that are set in cmask are cleared
in the mode of the created file.

RETURN VALUE
The file permission bits in the value returned by umask() will be the previous value of the file
mode creation mask. The state of any other bits in that value is unspecified, except that a
subsequent call to umask() with the returned value as cmask will leave the state of the mask the
same as its state before the first call, including any unspecified use of those bits.

ERRORS
No errors are defined.

SEE ALSO
creat(), mkdir(), mkfifo(), open(), <sys/stat.h>, <sys/types.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:
• The header <sys/types.h> is now marked as optional (oh); this header need not be included
on XSI-conformant systems.
• The RETURN VALUE section is expanded, in line with the ISO POSIX-1 standard, to
describe the situation with regard to additional bits in the file mode creation mask.
uname()  BASE  System Interfaces

NAME
uname — get name of current system

SYNOPSIS
#include <sys/utsname.h>
int uname(struct utsname *name);

DESCRIPTION
The uname() function stores information identifying the current system in the structure pointed to by name.

The uname() function uses the utsname structure defined in <sys/utsname.h>.

The uname() function returns a string naming the current system in the character array sysname. Similarly, nodename contains the name that the system is known by on a communications network. The arrays release and version further identify the operating system. The array machine contains a name that identifies the hardware that the system is running on.

The format of each member is implementation-dependent.

RETURN VALUE
Upon successful completion, a non-negative value is returned. Otherwise, –1 is returned and errno is set to indicate the error.

ERRORS
No errors are defined.

APPLICATION USAGE
The inclusion of the nodename member in this structure does not imply that it is sufficient information for interfacing to communications networks.

SEE ALSO
<sys/utsname.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated for alignment with the ISO POSIX-1 standard:

• The DESCRIPTION section is changed to indice that the format of members in the utsname structure is implementation-dependent.

• The RETURN VALUE section is updated to indicate that –1 will be returned and errno set to indicate an error.
NAME
ungetc — push byte back into input stream

SYNOPSIS
#include <stdio.h>

int ungetc(int c, FILE *stream);

DESCRIPTION
The ungetc() function pushes the byte specified by c (converted to an unsigned char) back onto
the input stream pointed to by stream. The pushed-back bytes will be returned by subsequent
reads on that stream in the reverse order of their pushing. A successful intervening call (with
the stream pointed to by stream) to a file-positioning function (fseek(), fsetpos() or rewind())
discards any pushed-back bytes for the stream. The external storage corresponding to the
stream is unchanged.

One byte of push-back is guaranteed. If ungetc() is called too many times on the same stream
without an intervening read or file-positioning operation on that stream, the operation may fail.
If the value of c equals that of the macro EOF, the operation fails and the input stream is
unchanged.

A successful call to ungetc() clears the end-of-file indicator for the stream. The value of the file-
position indicator for the stream after reading or discarding all pushed-back bytes will be the
same as it was before the bytes were pushed back. The file-position indicator is decremented by
each successful call to ungetc(); if its value was 0 before a call, its value is indeterminate after the
call.

RETURN VALUE
Upon successful completion, ungetc() returns the byte pushed back after conversion. Otherwise
it returns EOF.

ERRORS
No errors are defined.

SEE ALSO
fseek(), getc(), fsetpos(), read(), rewind(), setbuf(), <stdio.h>.

CHANGE HISTORY
First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated for alignment with the ISO C standard:
• The fsetpos() function is added to the list of file-positioning functions in the DESCRIPTION
  section.
• Also this issue states that the file-position indicator is decremented by each successful call to
  ungetc(), although note that XSI-conformant systems do not distinguish between text and
  binary streams. Previous issues state that the disposition of this indicator is unspecified.

Other changes are incorporated as follows:
• The DESCRIPTION is changed to make it clear that ungetc() manipulates bytes rather than
  (possibly multi-byte) characters.
• The APPLICATION USAGE section is removed.
ungetwc() — push wide-character code back into input stream

SYNOPSIS

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

wint_t ungetwc(wint_t wc, FILE *stream);
```

DESCRIPTION

The `ungetwc()` function pushes the character corresponding to the wide character code specified by `wc` back onto the input stream pointed to by `stream`. The pushed-back characters will be returned by subsequent reads on that stream in the reverse order of their pushing. A successful intervening call (with the stream pointed to by `stream`) to a file-positioning function (`fseek()`, `fsetpos()` or `rewind()`) discards any pushed-back characters for the stream. The external storage corresponding to the stream is unchanged.

One character of push-back is guaranteed. If `ungetwc()` is called too many times on the same stream without an intervening read or file-positioning operation on that stream, the operation may fail.

If the value of `wc` equals that of the macro `WEOF`, the operation fails and the input stream is unchanged.

A successful call to `ungetwc()` clears the end-of-file indicator for the stream. The value of the file-position indicator for the stream after reading or discarding all pushed-back characters will be the same as it was before the characters were pushed back. The file-position indicator is decremented (by one or more) by each successful call to `ungetwc()`; if its value was 0 before a call, its value is indeterminate after the call.

RETURN VALUE

Upon successful completion, `ungetwc()` returns the wide-character code corresponding to the pushed-back character. Otherwise it returns WEOF.

ERRORS

The `ungetwc()` function may fail if:

- `EILSEQ` An invalid character sequence is detected, or a wide-character code does not correspond to a valid character.

SEE ALSO

`fseek()`, `fsetpos()`, `read()`, `rewind()`, `setbuf()`, `<stdio.h>`, `<wchar.h>`.

CHANGE HISTORY

First released in Issue 4.

Derived from the MSE working draft.
NAME
unlink — remove directory entry

SYNOPSIS
#include <unistd.h>
int unlink(const char * path);

DESCRIPTION
UX The unlink() function removes a link to a file. If path names a symbolic link, unlink() removes the symbolic link named by path and does not affect any file or directory named by the contents of the symbolic link. Otherwise, unlink() removes the link named by the pathname pointed to by path and decrements the link count of the file referenced by the link.

When the file's link count becomes 0 and no process has the file open, the space occupied by the file will be freed and the file will no longer be accessible. If one or more processes have the file open when the last link is removed, the link will be removed before unlink() returns, but the removal of the file contents will be postponed until all references to the file are closed.

The path argument must not name a directory unless the process has appropriate privileges and the implementation supports using unlink() on directories.

Upon successful completion, unlink() will mark for update the st_ctime and st_mtime fields of the parent directory. Also, if the file's link count is not 0, the st_ctime field of the file will be marked for update.

RETURN VALUE
Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to indicate the error. If −1 is returned, the named file will not be changed.

ERRORS
The unlink() function will fail and not unlink the file if:

[EACCES] Search permission is denied for a component of the path prefix, or write permission is denied on the directory containing the directory entry to be removed.

[EBUSY] The file named by the path argument cannot be unlinked because it is being used by the system or another process and the implementation considers this an error, or the file named by path is a named STREAM.

[ELOOP] Too many symbolic links were encountered in resolving path.

[ENAMETOOLONG] The length of the path argument exceeds PATH_MAX or a pathname component is longer than NAME_MAX.

[ENOENT] A component of path does not name an existing file or path is an empty string.

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

[EPERM] The file named by path is a directory, and either the calling process does not have appropriate privileges, or the implementation prohibits using unlink() on directories.

UX [EPERM] or [EACCES] The S_ISVTX flag is set on the directory containing the file referred to by the path argument and the caller is not the file owner, nor is the caller the directory owner, nor does the caller have appropriate privileges.
unlink()

The directory entry to be unlinked is part of a read-only file system.

The `unlink()` function may fail and not unlink the file if:

<table>
<thead>
<tr>
<th>UX</th>
<th>[ENAMETOOLONG]</th>
</tr>
</thead>
<tbody>
<tr>
<td>UX</td>
<td>Pathname resolution of a symbolic link produced an intermediate result whose length exceeds [PATH_MAX].</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EX</th>
<th>[ETXTBSY]</th>
</tr>
</thead>
<tbody>
<tr>
<td>EX</td>
<td>The entry to be unlinked is the last directory entry to a pure procedure (shared text) file that is being executed.</td>
</tr>
</tbody>
</table>

**APPLICATION USAGE**

Applications should use `rmdir()` to remove a directory.

**SEE ALSO**

`close()`, `link()`, `remove()`, `rmdir()`, `<unistd.h>`.

**CHANGE HISTORY**

First released in Issue 1.

Derived from Issue 1 of the SVID.

**Issue 4**

The following change is incorporated for alignment with the ISO POSIX-1 standard:

- The type of argument `path` is changed from `char *` to `const char *`.

The following change is incorporated for alignment with the FIPS requirements:

- In the **ERRORS** section, the condition whereby [ENAMETOOLONG] will be returned if a pathname component is larger that [NAME_MAX] is now defined as mandatory and marked as an extension.

Other changes are incorporated as follows:

- The header `<unistd.h>` is added to the **SYNOPSIS** section.
- The error [ETXTBSY] is marked as an extension.

**Issue 4, Version 2**

The entry is updated for X/OPEN UNIX conformance as follows:

- In the **DESCRIPTION**, the effect is specified if `path` specifies a symbolic link.
- In the **ERRORS** section, [ELOOP] is added to indicate that too many symbolic links were encountered during pathname resolution.
- In the **ERRORS** section, [EPERM] or [EACCES] are added to indicate a permission check failure when operating on directories with S_ISVTX set.
- In the **ERRORS** section, a second [ENAMETOOLONG] condition is defined that may report excessive length of an intermediate result of pathname resolution of a symbolic link.
NAME
unlockpt — unlock a pseudo-terminal master/slave pair

SYNOPSIS
UX
#include <stdlib.h>

int unlockpt(int fildes);

DESCRIPTION
The unlockpt() function unlocks the slave pseudo-terminal device associated with the master to which fildes refers.

Portable applications must call unlockpt() before opening the slave side of a pseudo-terminal device.

RETURN VALUE
Upon successful completion, unlockpt() returns 0. Otherwise, it returns −1 and sets errno to indicate the error.

ERRORS
The unlockpt() function may fail if:

[EBADF] The fildes argument is not a file descriptor open for writing.

[EINVAL] The fildes argument is not associated with a master pseudo-terminal device.

SEE ALSO
grantpt(), open(), ptsname(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
usleep — suspend execution for an interval

SYNOPSIS

```c
#include <unistd.h>

int usleep(useconds_t useconds);
```

DESCRIPTION

The `usleep()` function suspends the current process from execution for the number of microseconds specified by the `useconds` argument. Because of other activity, or because of the time spent in processing the call, the actual suspension time may be longer than the amount of time specified.

The `useconds` argument must be less than 1,000,000. If the value of `useconds` is 0, then the call has no effect.

The `usleep()` function uses the process’ real-time interval timer to indicate to the system when the process should be woken up.

There is one real-time interval timer for each process. The `usleep()` function will not interfere with a previous setting of this timer. If the process has set this timer prior to calling `usleep()`, and if the time specified by `useconds` equals or exceeds the interval timer’s prior setting, the process will be woken up shortly before the timer was set to expire.

Implementations may place limitations on the granularity of timer values. For each interval timer, if the requested timer value requires a finer granularity than the implementation supports, the actual timer value will be rounded up to the next supported value.

Interactions between `usleep()` and either `alarm()` or `sleep()` are unspecified.

RETURN VALUE

On successful completion, `usleep()` returns 0. Otherwise, it returns −1 and sets `errno` to indicate the error.

ERRORS

The `usleep()` function may fail if:

- **[EINVAL]** The time interval specified 1,000,000 or more microseconds.

APPLICATION USAGE

The `usleep()` function is included for its historical usage. The `setitimer()` function is preferred over this function.

SEE ALSO

- `alarm()`, `getitimer()`, `sigaction()`, `sleep()`, `<unistd.h>`.  

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
    utime — set file access and modification times

SYNOPSIS
    #include <sys/types.h>
    #include <utime.h>
    int utime(const char *path, const struct utimbuf *times);

DESCRIPTION
    The utime() function sets the access and modification times of the file named by the path argument.

    If times is a null pointer, the access and modification times of the file are set to the current time.
    The effective user ID of the process must match the owner of the file, or the process must have write permission to the file or have appropriate privileges, to use utime() in this manner.

    If times is not a null pointer, times is interpreted as a pointer to a utimbuf structure and the access and modification times are set to the values contained in the designated structure. Only a process with effective user ID equal to the user ID of the file or a process with appropriate privileges may use utime() this way.

    The utimbuf structure is defined by the header <utime.h>. The times in the structure utimbuf are measured in seconds since the Epoch.

    Upon successful completion, utime() will mark the time of the last file status change, st_ctime, to be updated, see <sys/stat.h>.

RETURN VALUE
    Upon successful completion, 0 is returned. Otherwise, -1 is returned and errno is set to indicate the error, and the file times will not be affected.

ERRORS
    The utime() function will fail if:

    [EACCES]    Search permission is denied by a component of the path prefix; or the times argument is a null pointer and the effective user ID of the process does not match the owner of the file and write access is denied.

    [ELOOP]      Too many symbolic links were encountered in resolving path.

    [ENAMETOOLONG]  The length of the path argument exceeds [PATH_MAX] or a pathname component is longer than [NAME_MAX].

    [ENOENT]     A component of path does not name an existing file or path is an empty string.

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

    [EPERM]      The times argument is not a null pointer and the calling process' effective user ID has write access to the file but does not match the owner of the file and the calling process does not have the appropriate privileges.

    [EROFS]      The file system containing the file is read-only.

    The utime() function may fail if:

    [ENAMETOOLONG]  Pathname resolution of a symbolic link produced an intermediate result whose length exceeds [PATH_MAX].
SEE ALSO
   <sys/types.h>, <utime.h>.

CHANGE HISTORY
   First released in Issue 1.
   Derived from Issue 1 of the SVID.

Issue 4
   The following change is incorporated for alignment with the ISO POSIX-1 standard:
      • The type of argument path is changed from char * to const char *, and times is changed from
        struct utimbuf* to const struct utimbuf*.
   The following change is incorporated for alignment with the FIPS requirements:
      • In the ERRORS section, the condition whereby [ENAMETOOLONG] will be returned if a
        pathname component is larger than {NAME_MAX} is now defined as mandatory and marked
        as an extension.

Another change is incorporated as follows:
      • The header <sys/types.h> is now marked as optional (0H); this header need not be included
        on XSI-conformant systems.

Issue 4, Version 2
   The ERRORS section is updated for X/OPEN UNIX conformance as follows:
      • It states that [ELOOP] will be returned if too many symbolic links are encountered during
        pathname resolution.
      • A second [ENAMETOOLONG] condition is defined that may report excessive length of an
        intermediate result of pathname resolution of a symbolic link.
NAME
   utimes — set file access and modification times

SYNOPSIS
   int utimes(const char *path, const struct timeval times[2]);

DESCRIPTION
   The utimes() function sets the access and modification times of the file pointed to by the path argument to the value of the times argument. The utimes() function allows time specifications accurate to the microsecond.

   For utimes(), the times argument is an array of timeval structures. The first array member represents the date and time of last access, and the second member represents the date and time of last modification. The times in the timeval structure are measured in seconds and microseconds since the Epoch, although rounding toward the nearest second may occur.

   If the times argument is a null pointer, the access and modification times of the file are set to the current time. The effective user ID of the process must be the same as the owner of the file, or must have write access to the file or appropriate privileges to use this call in this manner. Upon completion, utimes() will mark the time of the last file status change, st_ctime, for update.

RETURN VALUE
   Upon successful completion, 0 is returned. Otherwise, -1 is returned and errno is set to indicate the error, and the file times will not be affected.

ERRORS
   The utimes() function will fail if:

   [EACCES] Search permission is denied by a component of the path prefix; or the times argument is a null pointer and the effective user ID of the process does not match the owner of the file and write access is denied.

   [ELOOP] Too many symbolic links were encountered in resolving path.

   [ENAMETOOLONG] The length of the path argument exceeds [PATH_MAX] or a pathname component is longer than [NAME_MAX].

   [ENOENT] A component of path does not name an existing file or path is an empty string.

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

   [EPERM] The times argument is not a null pointer and the calling process' effective user ID has write access to the file but does not match the owner of the file and the calling process does not have the appropriate privileges.

   [EROFS] The file system containing the file is read-only.

   The utimes() function may fail if:

   [ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds [PATH_MAX].

SEE ALSO
   <sys/time.h>.

CHANGE HISTORY
   First released in Issue 4, Version 2.
valloc()  X/OPEN UNIX  System Interfaces

NAME
valloc — page-aligned memory allocator (TO BE WITHDRAWN)

SYNOPSIS
UX
#include <stdlib.h>

void *valloc(size_t size);

DESCRIPTION
The valloc() function has the same effect as malloc(), except that the allocated memory will be
aligned to a multiple of the value returned by sysconf(_SC_PAGESIZE).

RETURN VALUE
Upon successful completion, valloc() returns a pointer to the allocated memory. Otherwise,
valloc() returns a null pointer and sets errno to indicate the error.

If size is 0, the behaviour is implementation-dependent; the value returned will be either a null
pointer or a unique pointer. When size is 0 and valloc() returns a null pointer, errno is not
modified.

ERRORS
The valloc() function will fail if:

[ENOMEM] Storage space available is insufficient.

APPLICATION USAGE
Applications should avoid using valloc() but should use malloc() or mmap() instead. On systems
with a large page size, the number of successful valloc() operations may be zero.

SEE ALSO
malloc(), sysconf(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME

vfork — create new process; share virtual memory

SYNOPSIS

UX

#include <unistd.h>

pid_t vfork(void);

DESCRIPTION

The vfork() function has the same effect as fork(), except that the behaviour is undefined if the process created by vfork() either modifies any data other than a variable of type pid_t used to store the return value from vfork(), or returns from the function in which vfork() was called, or calls any other function before successfully calling _exit() or one of the exec family of functions.

RETURN VALUE

Upon successful completion, vfork() returns 0 to the child process and returns the process ID of the child process to the parent process. Otherwise, −1 is returned to the parent, no child process is created, and errno is set to indicate the error.

ERRORS

The vfork() function will fail if:

[EAGAIN] The system-wide limit on the total number of processes under execution would be exceeded, or the system-imposed limit on the total number of processes under execution by a single user would be exceeded.

[ENOMEM] There is insufficient swap space for the new process.

APPLICATION USAGE

On some systems, vfork() is the same as fork().

The vfork() function differs from fork() only in that the child process can share code and data with the calling process (parent process). This speeds cloning activity significantly at a risk to the integrity of the parent process if vfork() is misused.

The use of vfork() for any purpose except as a prelude to an immediate call to a function from the exec family, or to _exit(), is not advised.

The vfork() function can be used to create new processes without fully copying the address space of the old process. If a forked process is simply going to call exec, the data space copied from the parent to the child by fork() is not used. This is particularly inefficient in a paged environment, making vfork() particularly useful. Depending upon the size of the parent’s data space, vfork() can give a significant performance improvement over fork().

The vfork() function can normally be used just like fork(). It does not work, however, to return while running in the child’s context from the caller of vfork() since the eventual return from vfork() would then return to a no longer existent stack frame. Be careful, also, to call _exit() rather than exit() if you cannot exec, since exit() flushes and closes standard I/O channels, thereby damaging the parent process’ standard I/O data structures. (Even with fork(), it is wrong to call exit(), since buffered data would then be flushed twice.)

If signal handlers are invoked in the child process after vfork(), they must follow the same rules as other code in the child process.

The [vfork, exec] window begins at the vfork() call and ends when the child completes its exec call.
SEE ALSO
exec, exit(), fork(), wait(), <unistd.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
vfprintf, vprintf, vsprintf — format output of a stdarg argument list

SYNOPSIS
#include <stdarg.h>
#include <stdio.h>

int vprintf(const char *format, va_list ap);
int vfprintf(FILE *stream, const char *format, va_list ap);
int vsprintf(char *s, const char *format, va_list ap);

DESCRIPTION
The vprintf(), vfprintf() and vsprintf() functions are the same as printf(), fprintf() and sprintf() respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by <stdarg.h>.

These functions do not invoke the va_end macro. As these functions invoke the va_arg macro, the value of ap after the return is indeterminate.

APPLICATION USAGE
Applications using these functions should call va_end(ap) afterwards to clean up.

RETURN VALUE
Refer to printf().

ERRORS
Refer to printf().

SEE ALSO
printf(), <stdarg.h>, <stdio.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated for alignment with the ISO C standard:

• These functions are no longer marked as extensions.
• The type of argument format is changed from char * to const char *.
• Reference to the <varargs.h> header in the DESCRIPTION section is replaced by <stdarg.h>.

The last paragraph has also been added to indicate interactions with the va_arg and va_end macros.

Other changes are incorporated as follows:

• The APPLICATION USAGE section is added.
• The FUTURE DIRECTIONS section is removed.
NAME

wait, waitpid — wait for child process to stop or terminate

SYNOPSIS

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *stat_loc);

pid_t waitpid(pid_t pid, int *stat_loc, int options);
```

DESCRIPTION

The `wait()` and `waitpid()` functions allow the calling process to obtain status information pertaining to one of its child processes. Various options permit status information to be obtained for child processes that have terminated or stopped. If status information is available for two or more child processes, the order in which their status is reported is unspecified.

The `wait()` function will suspend execution of the calling process until status information for one of its terminated child processes is available, or until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process. If status information is available prior to the call to `wait()`, return will be immediate.

The `waitpid()` function will behave identically to `wait()`, if the `pid` argument is `(pid_t)-1` and the `options` argument is 0. Otherwise, its behaviour will be modified by the values of the `pid` and `options` arguments.

The `pid` argument specifies a set of child processes for which status is requested. The `waitpid()` function will only return the status of a child process from this set:

- If `pid` is equal to `(pid_t)-1`, status is requested for any child process. In this respect, `waitpid()` is then equivalent to `wait()`.
- If `pid` is greater than 0, it specifies the process ID of a single child process for which status is requested.
- If `pid` is 0, status is requested for any child process whose process group ID is equal to that of the calling process.
- If `pid` is less than `(pid_t)-1`, status is requested for any child process whose process group ID is equal to the absolute value of `pid`.

The `options` argument is constructed from the bitwise-inclusive OR of zero or more of the following flags, defined in the header `<sys/wait.h>`.

- `WCONTINUED` The `waitpid()` function will report the status of any continued child process specified by `pid` whose status has not been reported since it continued from a job control stop.
- `WNOHANG` The `waitpid()` function will not suspend execution of the calling process if status is not immediately available for one of the child processes specified by `pid`.
- `WUNTRACED` The status of any child processes specified by `pid` that are stopped, and whose status has not yet been reported since they stopped, will also be reported to the requesting process.

If the calling process has `SA_NOCLDWAIT` set or has `SIGCHLD` set to `SIG_IGN`, and the process has no unwaited for children that were transformed into zombie processes, it will block until all of its children terminate, and `wait()` and `waitpid()` will fail and set `errno` to `[ECHILD]`. 

700 X/Open CAE Specification (1994)
If `wait()` or `waitpid()` return because the status of a child process is available, these functions will return a value equal to the process ID of the child process. In this case, if the value of the argument `stat_loc` is not a null pointer, information will be stored in the location pointed to by `stat_loc`. If and only if the status returned is from a terminated child process that returned 0 from `main()` or passed 0 as the `status` argument to `_exit()` or `exit()`, the value stored at the location pointed to by `stat_loc` will be 0. Regardless of its value, this information may be interpreted using the following macros, which are defined in `<sys/wait.h>` and evaluate to integral expressions; the `stat_val` argument is the integer value pointed to by `stat_loc`.

- **WIFEXITED(`stat_val`)** Evaluates to a non-zero value if status was returned for a child process that terminated normally.
- **WEXITSTATUS(`stat_val`)** If the value of WIFEXITED(`stat_val`) is non-zero, this macro evaluates to the low-order 8 bits of the `status` argument that the child process passed to `_exit()` or `exit()`, or the value the child process returned from `main()`.
- **WIFSIGNALED(`stat_val`)** Evaluates to non-zero value if status was returned for a child process that terminated due to the receipt of a signal that was not caught (see `<signal.h>`).
- **WTERMSIG(`stat_val`)** If the value of WIFSIGNALED(`stat_val`) is non-zero, this macro evaluates to the number of the signal that caused the termination of the child process.
- **WIFSTOPPED(`stat_val`)** Evaluates to a non-zero value if status was returned for a child process that is currently stopped.
- **WSTOPSIG(`stat_val`)** If the value of WIFSTOPPED(`stat_val`) is non-zero, this macro evaluates to the number of the signal that caused the child process to stop.
- **WIFCONTINUED(`stat_val`)** Evaluates to a non-zero value if status was returned for a child process that has continued from a job control stop.

If the information pointed to by `stat_loc` was stored by a call to `waitpid()` that specified the WUNTRACED flag and did not specify the WCONTINUED flag, exactly one of the macros WIFEXITED(`*stat_loc`), WIFSIGNALED(`*stat_loc`), and WIFSTOPPED(`*stat_loc`), will evaluate to a non-zero value.

If the information pointed to by `stat_loc` was stored by a call to `waitpid()` that specified the WUNTRACED and WCONTINUED flags, exactly one of the macros WIFEXITED(`*stat_loc`), WIFSIGNALED(`*stat_loc`), WIFSTOPPED(`*stat_loc`), and WIFCONTINUED(`*stat_loc`), will evaluate to a non-zero value.

If the information pointed to by `stat_loc` was stored by a call to `wait()` that did not specify the WUNTRACED or WCONTINUED flags, or by a call to the `wait()` function, exactly one of the macros WIFEXITED(`*stat_loc`) and WIFSIGNALED(`*stat_loc`) will evaluate to a non-zero value.

If the information pointed to by `stat_loc` was stored by a call to `waitpid()` that did not specify the WUNTRACED flag and specified the WCONTINUED flag, or by a call to the `wait()` function, exactly one of the macros WIFEXITED(`*stat_loc`), WIFSIGNALED(`*stat_loc`), and WIFCONTINUED(`*stat_loc`), will evaluate to a non-zero value.

There may be additional implementation-dependent circumstances under which `wait()` or `waitpid()` report status. This will not occur unless the calling process or one of its child processes explicitly makes use of a non-standard extension. In these cases the interpretation of the...
reported status is implementation-dependent.

If a parent process terminates without waiting for all of its child processes to terminate, the remaining child processes will be assigned a new parent process ID corresponding to an implementation-dependent system process.

**RETURN VALUE**

If `wait()` or `waitpid()` returns because the status of a child process is available, these functions will return a value equal to the process ID of the child process for which status is reported. If `wait()` or `waitpid()` returns due to the delivery of a signal to the calling process, −1 will be returned and `errno` will be set to `[EINTR]`. If `waitpid()` was invoked with `WNOHANG` set in `options`, it has at least one child process specified by `pid` for which status is not available, and status is not available for any process specified by `pid`, 0 will be returned. Otherwise, `(pid_t)−1` will be returned, and `errno` will be set to indicate the error.

**ERRORS**

The `wait()` function will fail if:

- `[ECHILD]` The calling process has no existing unwaited-for child processes.
- `[EINVAL]` The `options` argument is not valid.

The `waitpid()` function will fail if:

- `[ECHILD]` The process or process group specified by `pid` does not exist or is not a child of the calling process.
- `[EINTR]` The function was interrupted by a signal. The value of the location pointed to by `stat_loc` is undefined.
- `[EINVAL]` The `options` argument is not valid.

**SEE ALSO**

`exec`, `exit()`, `fork()`, `wait3()`, `waitid()`, `<sys/types.h>`, `<sys/wait.h>`.

**CHANGE HISTORY**

First released in Issue 1.

Derived from Issue 1 of the SVID.

**Issue 4**

The following change is incorporated for alignment with the ISO POSIX-1 standard:

- Text describing conditions under which 0 will be returned when `WNOHANG` is set in `options` is added to the RETURN VALUE section.

Other changes are incorporated as follows:

- The header `<sys/types.h>` is now marked as optional (`oh`); this header need not be included on XSI-conformant systems.
- Error return values throughout the DESCRIPTION and RETURN VALUE sections are changed to show the proper casting (that is, `(pid_t)−1`).
- The words "If the implementation supports job control” are removed from the description of `WUNTRACED`. This is because job control is defined as mandatory for Issue 4 conforming implementations.
Issues 4, Version 2

The following changes are incorporated in the DESCRIPTION for X/OPEN UNIX conformance:

- The WCONTINUED options flag and the WIFCONTINUED(stat_val) macro are added.
- Text following the list of options flags explains the implications of setting the SA_NOCLDWAIT signal flag, or setting SIGCHILD to SIG_IGN.
- Text following the list of macros, which explains what macros return non-zero values in certain cases, is expanded and the value of the WCONTINUED flag on the previous call to waitpid() is taken into account.
NAME
wait3 — wait for child process to change state

SYNOPSIS
UX
#include <sys/wait.h>

pid_t wait3 (int *stat_loc, int options, struct rusage *resource_usage);

DESCRIPTION
The wait3() function allows the calling process to obtain status information for specified child processes.

The following call:

wait3 (stat_loc, options, resource_usage);

is equivalent to the call:

waitpid ((pid_t)-1, stat_loc, options);

except that on successful completion, if the resource_usage argument to wait3() is not a null pointer, the rusage structure that the third argument points to is filled in for the child process identified by the return value.

RETURN VALUE
See waitpid().

ERRORS
In addition to the error conditions specified on waitpid(), under the following conditions, wait3() may fail and set errno to:

[ECHILD] The calling process has no existing unwaited-for child processes, or if the set of processes specified by the argument pid can never be in the states specified by the argument options.

SEE ALSO
exec, exit(), fork(), pause(), <sys/wait.h>.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME

waitid — wait for child process to change state

SYNOPSIS

UX

```c
#include <sys/wait.h>

int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);
```

DESCRIPTION

The `waitid()` function suspends the calling process until one of its children changes state. It records the current state of a child in the structure pointed to by `infop`. If a child process changed state prior to the call to `waitid()`, `waitid()` returns immediately.

The `idtype` and `id` arguments are used to specify which children `waitid()` will wait for.

If `idtype` is `P_PID`, `waitid()` will wait for the child with a process ID equal to `(pid_t)pid`.

If `idtype` is `P_PGID`, `waitid()` will wait for any child with a process group ID equal to `(pid_t)pid`.

If `idtype` is `P_ALL`, `waitid()` will wait for any children and `id` is ignored.

The `options` argument is used to specify which state changes `waitid()` will wait for. It is formed by OR-ing together one or more of the following flags:

- `WEXITED`: Wait for processes that have exited.
- `WSTOPPED`: Status will be returned for any child that has stopped upon receipt of a signal.
- `WCONTINUED`: Status will be returned for any child that was stopped and has been continued.
- `WNOHANG`: Return immediately if there are no children to wait for.
- `WNOWAIT`: Keep the process whose status is returned in `infop` in a waitable state. This will not affect the state of the process; the process may be waited for again after this call completes.

The `infop` argument must point to a `siginfo_t` structure. If `waitid()` returns because a child process was found that satisfied the conditions indicated by the arguments `idtype` and `options`, then the structure pointed to by `infop` will be filled in by the system with the status of the process. The `si_signo` member will always be equal to `SIGCHLD`.

RETURN VALUE

If `waitid()` returns due to the change of state of one of its children, 0 is returned. Otherwise, −1 is returned and `errno` is set to indicate the error.

ERRORS

The `waitid()` function will fail if:

- `[ECHILD]`: The calling process has no existing unwaited-for child processes.
- `[EINVAL]`: An invalid value was specified for `options`, or `idtype` and `id` specify an invalid set of processes.
SEE ALSO
   exec, exit(), wait(), <sys/wait.h>.

CHANGE HISTORY
   First released in Issue 4, Version 2.
NAME
waitpid — wait for child process to stop or terminate

SYNOPSIS

```c
#include <sys/types.h>
#include <sys/wait.h>

pid_t waitpid(pid_t pid, int *stat_loc, int options);
```

DESCRIPTION
Refer to `wait()`.

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
   wcscat — concatenate two wide character strings

SYNOPSIS
   WP
   #include <wchar.h>
   
   wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);

DESCRIPTION
   The wcscat() function appends a copy of the wide character string pointed to by ws2 (including
   the terminating null wide-character code) to the end of the wide character string pointed to by
   ws1. The initial wide-character code of ws2 overwrites the null wide-character code at the end of
   ws1. If copying takes place between objects that overlap, the behaviour is undefined.

RETURN VALUE
   The wcscat() function returns s1; no return value is reserved to indicate an error.

ERRORS
   No errors are defined.

SEE ALSO
   wcsncat(), <wchar.h>.

CHANGE HISTORY
   First released in Issue 4.
   Derived from the MSE working draft.
NAME
wcschr — wide character string scanning operation

SYNOPSIS
WP
#include <wchar.h>

wchar_t *wcschr(const wchar_t *ws, wchar_t wc);

DESCRIPTION
The wcschr() function locates the first occurrence of wc in the wide character string pointed to by
ws. The value of wc must be a character representable as a type wchar_t and must be a wide-
character code corresponding to a valid character in the current locale. The terminating null
wide-character code is considered to be part of the wide character string.

RETURN VALUE
Upon completion, wcschr() returns a pointer to the wide-character code, or a null pointer if the
wide-character code is not found.

ERRORS
No errors are defined.

SEE ALSO
wcsrchr(), <wchar.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
wcscmp — compare two wide character strings

SYNOPSIS

WP
#include <wchar.h>

int wcscmp(const wchar_t *ws1, const wchar_t *ws2);

DESCRIPTION
The wcscmp() function compares the wide character string pointed to by ws1 to the wide character string pointed to by ws2.
The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of wide-character codes that differ in the objects being compared.

RETURN VALUE
Upon completion, wcscmp() returns an integer greater than, equal to or less than 0, if the wide character string pointed to by ws1 is greater than, equal to or less than the wide character string pointed to by ws2 respectively.

ERRORS
No errors are defined.

SEE ALSO
wcscmp(), <wchar.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
wcscoll — wide character string comparison using collating information

SYNOPSIS
EI WP
#include <wchar.h>

int wcscoll(const wchar_t * ws1, const wchar_t * ws2);

DESCRIPTION
The wcscoll() function compares the wide character string pointed to by ws1 to the wide character string pointed to by ws2, both interpreted as appropriate to the LC_COLLATE category of the current locale.

RETURN VALUE
Upon successful completion, wcscoll() returns an integer greater than, equal to or less than 0, according to whether the wide character string pointed to by ws1 is greater than, equal to or less than the wide character string pointed to by ws2, when both are interpreted as appropriate to the current locale. On error, wcscoll() may set errno, but no return value is reserved to indicate an error.

ERRORS
The wcscoll() function may fail if:

[EINVAL] The ws1 or ws2 arguments contain wide character codes outside the domain of the collating sequence.
[ENOSYS] The function is not supported.

APPLICATION USAGE
Because no return value is reserved to indicate an error, an application wishing to check for error situations should set errno to 0, then call wcscoll(), then check errno and if it is non-zero, assume an error has occurred.

The wcsxfrm() and wcscmp() functions should be used for sorting large lists.

SEE ALSO
wcscmp(), wcsxfrm(), <wchar.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
wcscpy — copy a wide character string

SYNOPSIS
WP
#include <wchar.h>

wchar_t *wcscpy(wchar_t *ws1, const wchar_t *ws2);  

DESCRIPTION
The wcscpy() function copies the wide character string pointed to by ws2 (including the terminating null wide-character code) into the array pointed to by ws1. If copying takes place between objects that overlap, the behaviour is undefined.

RETURN VALUE
The wcscpy() function returns ws1; no return value is reserved to indicate an error.

ERRORS
No errors are defined.

APPLICATION USAGE
Wide character code movement is performed differently in different implementations. Thus overlapping moves may yield surprises.

SEE ALSO
wcscncpy(), <wchar.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
wcscspn — get length of complementary wide substring

SYNOPSIS
#include <wchar.h>
size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2);

DESCRIPTION
The wcscspn() function computes the length of the maximum initial segment of the wide character string pointed to by ws1 which consists entirely of wide-character codes not from the wide character string pointed to by ws2.

RETURN VALUE
The wcscspn() function returns ws1; no return value is reserved to indicate an error.

ERRORS
No errors are defined.

SEE ALSO
wcsspn( ), <wchar.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
wcsftime — convert date and time to wide character string

SYNOPSIS

```c
#include <wchar.h>

size_t wcsftime(wchar_t * wcs , size_t maxsize , const char * format ,
                const struct tm * timptr);
```

DESCRIPTION
The `wcsftime()` function places wide-character codes into the array pointed to by `wcs` as
controlled by the string pointed to by `format`.

This function behaves as if the character string generated by `strftime()` is passed to `mbstowcs()` as
the character string argument, and `mbstowcs()` places the result in the wide character string
argument of `wcsftime()` up to a limit of `maxsize` wide-character codes.

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

RETURN VALUE
If the total number of resulting wide character codes including the terminating null wide-
character code is no more than `maxsize`, `wcsftime()` returns the number of wide-character codes
placed into the array pointed to by `wcs`, not including the terminating null wide-character code.
Otherwise 0 is returned and the contents of the array are indeterminate. If the function is not
implemented, `errno` will be set to indicate the error.

ERRORS
The `wcsftime()` function will fail if:

- `[ENOSYS]` The function is not implemented.

SEE ALSO

- `strftime()`, `mbstowcs()`, `<wchar.h>`.

CHANGE HISTORY
First released in Issue 4.
Derived from the ISO SC22/WG14/N104 draft.
NAME
wcslen — get wide character string length

SYNOPSIS

```c
#include <wchar.h>

size_t wcslen(const wchar_t *ws);
```

DESCRIPTION
The `wcslen()` function computes the number of wide-character codes in the wide character string to which `ws` points, not including the terminating null wide-character code.

RETURN VALUE
The `wcslen()` function returns `ws`; no return value is reserved to indicate an error.

ERRORS
No errors are defined.

SEE ALSO
`<wchar.h>`.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
wcsncat — concatenate part of two wide character strings

SYNOPSIS

```c
#include <wchar.h>

wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
```

DESCRIPTION
The wcsncat() function appends not more than \( n \) wide-character codes (a null wide-character code and wide character codes that follow it are not appended) from the array pointed to by \( ws2 \) to the end of the wide character string pointed to by \( ws1 \). The initial wide-character code of \( ws2 \) overwrites the null wide-character code at the end of \( ws1 \). A terminating null wide-character code is always appended to the result. If copying takes place between objects that overlap, the behaviour is undefined.

RETURN VALUE
The wcsncat() function returns \( ws1 \); no return value is reserved to indicate an error.

ERRORS
No errors are defined.

SEE ALSO
wcsat(), wchar.h.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
wcsncmp — compare part of two wide character strings

SYNOPSIS
WP

```c
#include <wchar.h>

int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
```

DESCRIPTION
The `wcsncmp()` function compares not more than `n` wide-character codes (wide-character codes that follow a null wide character code are not compared) from the array pointed to by `ws1` to the array pointed to by `ws2`.

The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of wide-character codes that differ in the objects being compared.

RETURN VALUE
Upon successful completion, `wcsncmp()` returns an integer greater than, equal to or less than 0, if the possibly null-terminated array pointed to by `ws1` is greater than, equal to or less than the possibly null-terminated array pointed to by `ws2` respectively.

ERRORS
No errors are defined.

SEE ALSO
`wcscmp()`, `<wchar.h>`.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME

wcsncpy — copy part of a wide character string

SYNOPSIS

```c
#include <wchar.h>

wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
```

DESCRIPTION

The `wcsncpy()` function copies not more than `n` wide-character codes (wide-character codes that follow a null wide character code are not copied) from the array pointed to by `ws2` to the array pointed to by `ws1`. If copying takes place between objects that overlap, the behaviour is undefined.

If the array pointed to by `ws2` is a wide character string that is shorter than `n` wide-character codes, null wide-character codes are appended to the copy in the array pointed to by `ws1`, until `n` wide-character codes in all are written.

RETURN VALUE

The `wcsncpy()` function returns `ws1`; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

APPLICATION USAGE

Wide character code movement is performed differently in different implementations. Thus overlapping moves may yield surprises.

If there is no null wide-character code in the first `n` wide-character codes of the array pointed to by `ws2`, the result will not be null-terminated.

SEE ALSO

`wcsncpy()`, `<wchar.h>`.

CHANGE HISTORY

First released in Issue 4.

Derived from the MSE working draft.
NAME
wcspbrk — scan wide character string for wide-character code

SYNOPSIS
#include <wchar.h>

wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);

DESCRIPTION
The wcspbrk() function locates the first occurrence in the wide character string pointed to by ws1
of any wide-character code from the wide character string pointed to by ws2.

RETURN VALUE
Upon successful completion, wcspbrk() returns a pointer to the wide-character code or a null
pointer if no wide-character code from ws2 occurs in ws1.

ERRORS
No errors are defined.

SEE ALSO
wcscchr(), wcsrchr(), <wchar.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
wcsrchr — wide character string scanning operation

SYNOPSIS
#include <wchar.h>

wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

DESCRIPTION
The wcsrchr() function locates the last occurrence of wc in the wide character string pointed to by ws. The value of wc must be a character representable as a type wchar_t and must be a wide-character code corresponding to a valid character in the current locale. The terminating null wide-character code is considered to be part of the wide character string.

RETURN VALUE
Upon successful completion, wcsrchr() returns a pointer to the wide-character code or a null pointer if wc does not occur in the wide character string.

ERRORS
No errors are defined.

SEE ALSO
wcschr(), <wchar.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
wcsspn — get length of wide substring

SYNOPSIS

```c
#include <wchar.h>

size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);
```

DESCRIPTION
The `wcsspn()` function computes the length of the maximum initial segment of the wide character string pointed to by `ws1` which consists entirely of wide-character codes from the wide string pointed to by `ws2`.

RETURN VALUE
The `wcsspn()` function returns `ws1`; no return value is reserved to indicate an error.

ERRORS
No errors are defined.

SEE ALSO
`wcscspn()`, `<wchar.h>`.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
wcstod — convert wide character string to double-precision number

SYNOPSIS
#include <wchar.h>

double wcstod(const wchar_t * nptr , wchar_t ** endptr );

DESCRIPTION
The wcstod() function converts the initial portion of the wide character string pointed to by nptr to double representation. First it decomposes the input wide character string into three parts: an initial, possibly empty, sequence of white-space wide character codes (as specified by iswspace()); a subject sequence interpreted as a floating-point constant; and a final wide-character string of one or more unrecognised wide-character codes, including the terminating null wide character code of the input wide character string. Then it attempts to convert the subject sequence to a floating-point number, and returns the result.

The expected form of the subject sequence is an optional + or – sign, then a non-empty sequence of digits optionally containing a radix, then an optional exponent part. An exponent part consists of e or E, followed by an optional sign, followed by one or more decimal digits. The subject sequence is defined as the longest initial subsequence of the input wide character string, starting with the first non-white-space wide-character code, that is of the expected form. The subject sequence contains no wide-character codes if the input wide character string is empty or consists entirely of white-space wide-character codes, or if the first wide-character code that is not white space other than a sign, a digit or a radix.

If the subject sequence has the expected form, the sequence of wide-character codes starting with the first digit or the radix (whichever occurs first) is interpreted as a floating constant as defined in the C language, except that the radix is used in place of a period, and that if neither an exponent part nor a radix appears, a radix is assumed to follow the last digit in the wide character string. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final wide character string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

The radix is defined in the program’s locale (category LC_NUMERIC). In the POSIX locale, or in a locale where the radix is not defined, the radix defaults to a period (.).

In other than the POSIX locale, other implementation-dependent subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of nptr is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

RETURN VALUE
The wcstod() function returns the converted value, if any. If no conversion could be performed, 0 is returned and errno may be set to [EINVAL].

If the correct value is outside the range of representable values, ±HUGE_VAL is returned (according to the sign of the value), and errno is set to [ERANGE].

If the correct value would cause underflow, 0 is returned and errno is set to [ERANGE].

ERRORS
The wcstod() function will fail if:
[ERANGE] The value to be returned would cause overflow or underflow.
The \texttt{wcstod()} function may fail if:

EX

\textbf{[EINVAL]} No conversion could be performed.

APPLICATION USAGE

Because 0 is returned on error and is also a valid return on success, an application wishing to check for error situations should set \texttt{errno} to 0, then call \texttt{wcstod()}, then check \texttt{errno} and if it is non-zero, assume an error has occurred.

SEE ALSO

\texttt{iswspace()}, \texttt{localeconv()}, \texttt{scanf()}, \texttt{setlocale()}, \texttt{wstol()}, \texttt{<wchar.h>}, the XBD specification, Chapter 5, Locale.

CHANGE HISTORY

First released in Issue 4.

Derived from the MSE working draft.
NAME
wcstok — split wide character string into tokens

SYNOPSIS
```
#include <wchar.h>

wchar_t *wcstok(wchar_t * ws1 , const wchar_t * ws2 );
```

DESCRIPTION
A sequence of calls to wcstok() breaks the wide character string pointed to by ws1 into a
sequence of tokens, each of which is delimited by a wide-character code from the wide character
string pointed to by ws2. The first call in the sequence has ws1 as its first argument, and is
followed by calls with a null pointer as their first argument. The separator string pointed to by
ws2 may be different from call to call.

The first call in the sequence searches the wide character string pointed to by ws1 for the first
wide-character code that is not contained in the current separator string pointed to by ws2. If no
such wide-character code is found, then there are no tokens in the wide character string pointed
to by ws1 and wcstok() returns a null pointer. If such a wide-character code is found, it is the
start of the first token.

The wcstok() function then searches from there for a wide-character code that is contained in the
current separator string. If no such wide-character code is found, the current token extends to
the end of the wide character string pointed to by ws1, and subsequent searches for a token will
return a null pointer. If such a wide-character code is found, it is overwritten by a null wide
character, which terminates the current token. The wcstok() function saves a pointer to the
following wide-character code, from which the next search for a token will start.

Each subsequent call, with a null pointer as the value of the first argument, starts searching from
the saved pointer and behaves as described above.

The implementation will behave as if no function calls wcstok().

RETURN VALUE
Upon successful completion, the wcstok() function returns a pointer to the first wide-character
code of a token. Otherwise, if there is no token, wcstok() returns a null pointer.

ERRORS
No errors are defined.

SEE ALSO
<wchar.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
wcstol — convert wide character string to long integer

SYNOPSIS

#include <wchar.h>

long int wcstol(const wchar_t *nptr, wchar_t **endptr, int base);

DESCRIPTION
The wcstol() function converts the initial portion of the wide character string pointed to by nptr to
long int representation. First it decomposes the input wide character string into three parts:
an initial, possibly empty, sequence of white-space wide-character codes (as specified by
iswspace()), a subject sequence interpreted as an integer represented in some radix determined
by the value of base; and a final wide character string of one or more unrecognised wide
character codes, including the terminating null wide-character code of the input wide character
string. Then it attempts to convert the subject sequence to an integer, and returns the result.

If base is 0, the expected form of the subject sequence is that of a decimal constant, octal constant
or hexadecimal constant, any of which may be preceded by a + or – sign. A decimal constant
begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant
consists of the prefix 0 optionally followed by a sequence of the digits 0 to 7 only. A
hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits
and letters a (or A) to f (or F) with values 10 to 15 respectively.

If the value of base is between 2 and 36, the expected form of the subject sequence is a sequence
of letters and digits representing an integer with the radix specified by base, optionally preceded
by a + or – sign, but not including an integer suffix. The letters from a (or A) to z (or Z) inclusive
are ascribed the values 10 to 35; only letters whose ascribed values are less than that of base
are permitted. If the value of base is 16, the wide-character code representations of 0x or 0X may
optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input wide character
string, starting with the first non-white-space wide-character code, that is of the expected form.
The subject sequence contains no wide-character codes if the input wide character string is
empty or consists entirely of white-space wide-character code, or if the first non-white-space
wide-character code is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and base is 0, the sequence of wide-character codes
starting with the first digit is interpreted as an integer constant. If the subject sequence has the
expected form and the value of base is between 2 and 36, it is used as the base for conversion,
assigning to each letter its value as given above. If the subject sequence begins with a minus
sign, the value resulting from the conversion is negated. A pointer to the final wide character
string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

In other than the POSIX locale, additional implementation-dependent subject sequence forms
may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed;
the value of nptr is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

RETURN VALUE
Upon successful completion, wcstol() returns the converted value, if any. If no conversion could
be performed, 0 is returned and errno may be set to indicate the error. If the correct value is
outside the range of representable values, {LONG_MAX} or {LONG_MIN} is returned
(according to the sign of the value), and errno is set to [ERANGE].
The `wcstol()` function will fail if:

- **[EINVAL]** The value of `base` is not supported.
- **[ERANGE]** The value to be returned is not representable.

The `wcstol()` function may fail if:

- **[EINVAL]** No conversion could be performed.

**APPLICATION USAGE**

Because 0, *[LONG_MIN]* and *[LONG_MAX]* are returned on error and are also valid returns on success, an application wishing to check for error situations should set `errno` to 0, then call `wcstol()`, then check `errno` and if it is 0, assume an error has occurred.

**SEE ALSO**

`iswalpha()`, `scanf()`, `w值守()`, `<wchar.h>`.

**CHANGE HISTORY**

First released in Issue 4.

Derived from the MSE working draft.
NAME
wcstombs — convert a wide character string to a character string

SYNOPSIS
#include <stdlib.h>

size_t wcstombs(char * s, const wchar_t * pwcs, size_t n);

DESCRIPTION
The wcstombs() function converts the sequence of wide-character codes that are in the array
pointed to by pwcs into a sequence of characters that begins in the initial shift state and stores
these characters into the array pointed to by s, stopping if a character would exceed the limit of n
total bytes or if a null byte is stored. Each wide-character code is converted as if by a call to
wctomb(), except that the shift state of wctomb() is not affected.

The behaviour of this function is affected by the LC_CTYPE category of the current locale.

No more than n bytes will be modified in the array pointed to by s. If copying takes place
between objects that overlap, the behaviour is undefined. If s is a null pointer, wcstombs() returns
the length required to convert the entire array regardless of the value of n, but no values are
stored. The array will not be null-terminated if the value returned is n.

RETURN VALUE
If a wide-character code is encountered that does not correspond to a valid character (of one or
more bytes each), wcstombs() returns (size_t)−1. Otherwise, wcstombs() returns the number of
bytes stored in the character array, not including any terminating null byte. The array will not
be null-terminated if the value returned is n.

ERRORS
The wcstombs() function may fail if:

EX [EILSEQ] A wide-character code does not correspond to a valid character.

SEE ALSO
mblen(), mbtowc(), mbstowcs(), wctomb(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the ISO C standard.
NAME
wcstoul — convert wide character string to unsigned long

SYNOPSIS

```c
#include <wchar.h>

unsigned long int wcstoul(const wchar_t * nptr , wchar_t * *endptr ,
    int base);
```

DESCRIPTION

The `wcstoul()` function converts the initial portion of the wide character string pointed to by `nptr` to `unsigned long int` representation. First it decomposes the input wide-character string into three parts: an initial, possibly empty, sequence of white-space wide-character codes (as specified by `iswspace()`); a subject sequence interpreted as an integer represented in some radix determined by the value of `base`; and a final wide-character string of one or more unrecognised wide character codes, including the terminating null wide-character code of the input wide character string. Then it attempts to convert the subject sequence to an unsigned integer, and returns the result.

If `base` is 0, the expected form of the subject sequence is that of a decimal constant, octal constant or hexadecimal constant, any of which may be preceded by a `+` or `−` sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix 0 optionally followed by a sequence of the digits 0 to 7 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters a (or A) to f (or F) with values 10 to 15 respectively.

If the value of `base` is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by `base`, optionally preceded by a `+` or `−` sign, but not including an integer suffix. The letters from a (or A) to z (or Z) inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of `base` are permitted. If the value of `base` is 16, the wide-character codes 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input wide-character string, starting with the first wide-character code that is not white space and is of the expected form. The subject sequence contains no wide-character codes if the input wide-character string is empty or consists entirely of white-space wide-character codes, or if the first wide-character code that is not white space is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and `base` is 0, the sequence of wide-character codes starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of `base` is between 2 and 36, it is used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final wide character string is stored in the object pointed to by `endptr`, provided that `endptr` is not a null pointer.

In other than the POSIX locale, additional implementation-dependent subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of `nptr` is stored in the object pointed to by `endptr`, provided that `endptr` is not a null pointer.
RETURN VALUE
Upon successful completion, \texttt{wcstoul()} returns the converted value, if any. If no conversion could be performed, 0 is returned and \texttt{errno} may be set to indicate the error. If the correct value is outside the range of representable values, \texttt{[ULONG_MAX]} is returned and \texttt{errno} is set to \texttt{[ERANGE]}.

ERRORS
The \texttt{wcstoul()} function will fail if:

- \texttt{[EINVAL]} The value of \texttt{base} is not supported.
- \texttt{[ERANGE]} The value to be returned is not representable.

The \texttt{wcstoul()} function may fail if:

- \texttt{[EINVAL]} No conversion could be performed.

APPLICATION USAGE
Because 0 and \texttt{[ULONG_MAX]} are returned on error and 0 is also a valid return on success, an application wishing to check for error situations should set \texttt{errno} to 0, then call \texttt{wcstoul()}, then check \texttt{errno} and if it is non-zero, assume an error has occurred.

Unlike \texttt{wcstod()} and \texttt{wcstol()}, \texttt{wcstoul()} must always return a non-negative number; so, using the return value of \texttt{wcstoul()} for out-of-range numbers with \texttt{wcstoul()} could cause more severe problems than just loss of precision if those numbers can ever be negative.

SEE ALSO
\texttt{iswalpha()}, \texttt{scanf()}, \texttt{wcstod()}, \texttt{wcstol()}, \texttt{<wchar.h>}.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
wcswcs — find wide substring

SYNOPSIS
WP
#include <wchar.h>

wchar_t *wcswcs(const wchar_t *ws1, const wchar_t *ws2);

DESCRIPTION
The wcswcs() function locates the first occurrence in the wide character string pointed to by ws1 of the sequence of wide-character codes (excluding the terminating null wide-character code) in the wide character string pointed to by ws2.

RETURN VALUE
Upon successful completion, wcswcs() returns a pointer to the located wide character string or a null pointer if the wide character string is not found.

If ws2 points to a wide character string with zero length, the function returns ws1.

ERRORS
No errors are defined.

SEE ALSO
wcscs(), <wchar.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
wcswidth — number of column positions of a wide character string

SYNOPSIS

```
#include <wchar.h>

int wcswidth(const wchar_t *pwcs, size_t n);
```

DESCRIPTION
The wcswidth() function determines the number of column positions required for \( n \) wide-character codes (or fewer than \( n \) wide-character codes if a null wide-character code is encountered before \( n \) wide-character codes are exhausted) in the string pointed to by \( pwcs \).

RETURN VALUE
The wcswidth() function either returns 0 (if \( pwcs \) points to a null wide-character code), or returns the number of column positions to be occupied by the wide character string pointed to by \( pwcs \), or returns −1 (if any of the first \( n \) wide-character codes in the wide character string pointed to by \( pwcs \) is not a printing wide-character code).

ERRORS
No errors are defined.

SEE ALSO
wcwidth(), <wchar.h>, the definition of Column Position in the XBD specification, Chapter 2, Glossary.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
**NAME**
wcsxfrm — wide character string transformation

**SYNOPSIS**
```c
#include <wchar.h>

size_t wcsxfrm(wchar_t * ws1, const wchar_t * ws2, size_t n);
```

**DESCRIPTION**
The `wcsxfrm()` function transforms the wide character string pointed to by `ws2` and places the resulting wide character string into the array pointed to by `ws1`. The transformation is such that if `wcscmp()` is applied to two transformed wide strings, it returns a value greater than, equal to or less than 0, corresponding to the result of `wcscoll()` applied to the same two original wide character strings. No more than `n` wide-character codes are placed into the resulting array pointed to by `ws1`, including the terminating null wide-character code. If `n` is 0, `ws1` is permitted to be a null pointer. If copying takes place between objects that overlap, the behaviour is undefined.

**RETURN VALUE**
The `wcsxfrm()` function returns the length of the transformed wide character string (not including the terminating null wide-character code). If the value returned is `n` or more, the contents of the array pointed to by `ws1` are indeterminate.

On error, the `wcsxfrm()` function returns `(size_t)-1`, and sets `errno` to indicate the error.

**ERRORS**
The `wcsxfrm()` function may fail if:

- **[EINVAL]** The wide character string pointed to by `ws2` contains wide-character codes outside the domain of the collating sequence.
- **[ENOSYS]** The function is not supported.

**APPLICATION USAGE**
The transformation function is such that two transformed wide character strings can be ordered by `wcscmp()` as appropriate to collating sequence information in the program’s locale (category `LC_COLLATE`).

The fact that when `n` is 0, `ws1` is permitted to be a null pointer, is useful to determine the size of the `ws1` array prior to making the transformation.

Because no return value is reserved to indicate an error, an application wishing to check for error situations should set `errno` to 0, then call `wcscoll()`, then check `errno` and if it is non-zero, assume an error has occurred.

**SEE ALSO**
`wcscmp()`, `wcscoll()`, `<wchar.h>`.

**CHANGE HISTORY**
First released in Issue 4.
Derived from the MSE working draft.
NAME
wctomb — convert a wide-character code to a character

SYNOPSIS
#include <stdlib.h>

int wctomb(char * s, wchar_t wchar);

DESCRIPTION
The wctomb() function determines the number of bytes needed to represent the character corresponding to the wide-character code whose value is wchar (including any change in the shift state). It stores the character representation (possibly multiple bytes and any special bytes to change shift state) in the array object pointed to by s (if s is not a null pointer). At most \{MB_CUR_MAX\} bytes are stored. If wchar is 0, wctomb() is left in the initial shift state.

The behaviour of this function is affected by the LC_CTYPE category of the current locale. For a state-dependent encoding, this function is placed into its initial state by a call for which its character pointer argument, s, is a null pointer. Subsequent calls with s as other than a null pointer cause the internal state of the function to be altered as necessary. A call with s as a null pointer causes this function to return a non-zero value if encodings have state dependency, and 0 otherwise. Changing the LC_CTYPE category causes the shift state of this function to be indeterminate.

The implementation will behave as if no function defined in this document calls wctomb().

RETURN VALUE
If s is a null pointer, wctomb() returns a non-zero or 0 value, if character encodings, respectively, do or do not have state-dependent encodings. If s is not a null pointer, wctomb() returns −1 if the value of wchar does not correspond to a valid character, or returns the number of bytes that constitute the character corresponding to the value of wchar.

In no case will the value returned be greater than the value of the MB_CUR_MAX macro.

ERRORS
No errors are defined.

SEE ALSO
mblen(), mbtowc(), mbstowcs(), wcstombs(), <stdlib.h>.

CHANGE HISTORY
First released in Issue 4.

Derived from the ANSI C standard.
NAME

wctype - define character class

SYNOPSIS

```
#include <wchar.h>

wctype_t wctype(const char * charclass);
```

DESCRIPTION

The `wctype()` function is defined for valid character class names as defined in the current locale. The `charclass` is a string identifying a generic character class for which codeset-specific type information is required. The following character class names are defined in all locales — "alnum", "alpha", "blank", "cntrl", "digit", "graph", "lower", "print", "punct", "space", "upper" and "xdigit".

Additional character class names defined in the locale definition file (category LC_CTYPE) can also be specified.

The function returns a value of type `wctype_t`, which can be used as the second argument to subsequent calls of `iswctype()`. The `wctype()` function determines values of `wctype_t` according to the rules of the coded character set defined by character type information in the program's locale (category LC_CTYPE). The values returned by `wctype()` are valid until a call to `setlocale()` that modifies the category LC_CTYPE.

RETURN VALUE

The `wctype()` function returns 0 if the given character class name is not valid for the current locale (category LC_CTYPE), otherwise it returns an object of type `wctype_t` that can be used in calls to `iswctype()`.

ERRORS

No errors are defined.

SEE ALSO

`iswctype()`, `<wchar.h>`.

CHANGE HISTORY

First released in Issue 4.

Derived from the MSE working draft.
NAME
wcwidth — number of column positions of a wide-character code

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

int wcwidth(wchar_t wc);
```

DESCRIPTION
The `wcwidth()` function determines the number of column positions required for the wide character `wc`. The value of `wc` must be a character representable as a `wchar_t`, and must be a wide-character code corresponding to a valid character in the current locale.

RETURN VALUE
The `wcwidth()` function either returns 0 (if `wc` is a null wide-character code), or returns the number of column positions to be occupied by the wide-character code `wc`, or returns −1 (if `wc` does not correspond to a printing wide-character code).

ERRORS
No errors are defined.

SEE ALSO
`wcswidth()`, `<wchar.h>`.

CHANGE HISTORY
First released as a World-wide Portability Interface in Issue 4.
Derived from MSE working draft.
NAME
wordexp, wordfree — perform word expansions

SYNOPSIS
#include <wordexp.h>

int wordexp(const char *words, wordexp_t *pwordexp, int flags);
void wordfree(wordexp_t *pwordexp);

DESCRIPTION
The wordexp() function performs word expansions as described in the XCU specification, Section 2.6, Word Expansions, subject to quoting as in the XCU specification, Section 2.2, Quoting, and places the list of expanded words into the structure pointed to by pwordexp.

The words argument is a pointer to a string containing one or more words to be expanded. The expansions will be the same as would be performed by the shell if words were the part of a command line representing the arguments to a utility. Therefore, words must not contain an unquoted newline or any of the unquoted shell special characters:

| & ; < > |

except in the context of command substitution as specified in the XCU specification, Section 2.6.3, Command Substitution. It also must not contain unquoted parentheses or braces, except in the context of command or variable substitution. If the argument words contains an unquoted comment character (number sign) that is the beginning of a token, wordexp() may treat the comment character as a regular character, or may interpret it as a comment indicator and ignore the remainder of words.

The structure type wordexp_t is defined in the header <wordexp.h> and includes at least the following members:

<table>
<thead>
<tr>
<th>Member Type</th>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_t</td>
<td>we_wordc</td>
<td>Count of words matched by words.</td>
</tr>
<tr>
<td>char **</td>
<td>we_wordv</td>
<td>Pointer to list of expanded words.</td>
</tr>
<tr>
<td>size_t</td>
<td>we_offs</td>
<td>Slots to reserve at the beginning of pwordexp-&gt;we_wordv.</td>
</tr>
</tbody>
</table>

The wordexp() function stores the number of generated words into pwordexp->we_wordc and a pointer to a list of pointers to words in pwordexp->we_wordv. Each individual field created during field splitting (see the XCU specification, Section 2.6.5, Field Splitting) or pathname expansion (see the XCU specification, Section 2.6.6, Pathname Expansion) is a separate word in the pwordexp->we_wordv list. The words are in order as described in the XCU specification, Section 2.6, Word Expansions. The first pointer after the last word pointer will be a null pointer.

The expansion of special parameters described in the XCU specification, Section 2.5.2, Special Parameters is unspecified.

It is the caller's responsibility to allocate the storage pointed to by pwordexp. The wordexp() function allocates other space as needed, including memory pointed to by pwordexp->we_wordv. The wordfree() function frees any memory associated with pwordexp from a previous call to wordexp().

The flags argument is used to control the behaviour of wordexp(). The value of flags is the bitwise inclusive OR of zero or more of the following constants, which are defined in <wordexp.h>:

- WRDE_APPEND: Append words generated to the ones from a previous call to wordexp().
- WRDE_DOOFFS: Make use of pwordexp->we_offs. If this flag is set, pwordexp->we_offs is used to specify how many null pointers to add to the beginning of
In other words, `pwordexp->we_wordv` will point to `pwordexp->we_offs` null pointers, followed by `pwordexp->we_wordc` word pointers, followed by a null pointer.

- **WRDE_NOCMD**: Fail if command substitution, as specified in the XCU specification, Section 2.6.3, Command Substitution, is requested.

- **WRDE_REUSE**: The `pwordexp` argument was passed to a previous successful call to `wordexp()`, and has not been passed to `wordfree()`. The result will be the same as if the application had called `wordfree()` and then called `wordexp()` without WRDE_REUSE.

- **WRDE_SHOWERR**: Do not redirect `stderr` to `/dev/null`.

- **WRDE_UNDEF**: Report error on an attempt to expand an undefined shell variable.

The WRDE_APPEND flag can be used to append a new set of words to those generated by a previous call to `wordexp()`. The following rules apply when two or more calls to `wordexp()` are made with the same value of `pwordexp` and without intervening calls to `wordfree()`:

1. The first such call must not set WRDE_APPEND. All subsequent calls must set it.
2. All of the calls must set WRDE_DOOFFS, or all must not set it.
3. After the second and each subsequent call, `pwordexp->we_wordv` will point to a list containing the following:
   a. zero or more null pointers, as specified by WRDE_DOOFFS and `pwordexp->we_offs`
   b. pointers to the words that were in the `pwordexp->we_wordv` list before the call, in the same order as before
   c. pointers to the new words generated by the latest call, in the specified order
4. The count returned in `pwordexp->we_wordc` will be the total number of words from all of the calls.
5. The application can change any of the fields after a call to `wordexp()`, but if it does it must reset them to the original value before a subsequent call, using the same `pwordexp` value, to `wordfree()` or `wordexp()` with the WRDE_APPEND or WRDE_REUSE flag.

If `words` contains an unquoted:

```plaintext
<newline> | & ; < > ( ) {}
```

in an inappropriate context, `wordexp()` will fail, and the number of expanded words will be 0.

Unless WRDE_SHOWERR is set in `flags`, `wordexp()` will redirect `stderr` to `/dev/null` for any utilities executed as a result of command substitution while expanding `words`. If WRDE_SHOWERR is set, `wordexp()` may write messages to `stderr` if syntax errors are detected while expanding `words`.

If WRDE_DOOFFS is set, then `pwordexp->we_offs` must have the same value for each `wordexp()` call and `wordfree()` call using a given `pwordexp`.

The following constants are defined as error return values:

- **WRDE_BADCHAR**: One of the unquoted characters:

```plaintext
<newline> | & ; < > ( ) {}
```

appears in `words` in an inappropriate context.
wordexp()  POSIX2 CLB

WRDE_BADVAL  Reference to undefined shell variable when WRDE_UNDEF is set in flags.
WRDE_CMDSUB  Command substitution requested when WRDE_NOCMD was set in flags.
WRDE_NOSPACE Attempt to allocate memory failed.
WRDE_SYNTAX  Shell syntax error, such as unbalanced parentheses or unterminated string.

RETURN VALUE
On successful completion, wordexp() returns 0.

Otherwise, a non-zero value as described in <wordexp.h> is returned to indicate an error. If wordexp() returns the value WRDE_NOSPACE, then pwordexp->we_wordc and pwordexp->we_wordv will be updated to reflect any words that were successfully expanded. In other cases, they will not be modified.

The wordfree() function returns no value.

ERRORS
No errors are defined.

APPLICATION USAGE
This function is intended to be used by an application that wants to do all of the shell’s expansions on a word or words obtained from a user. For example, if the application prompts for a filename (or list of filenames) and then uses wordexp() to process the input, the user could respond with anything that would be valid as input to the shell.

The WRDE_NOCMD flag is provided for applications that, for security or other reasons, want to prevent a user from executing shell commands. Disallowing unquoted shell special characters also prevents unwanted side effects such as executing a command or writing a file.

SEE ALSO
fnmatch(), glob(), <wordexp.h>, the XCU specification.

CHANGE HISTORY
First released in Issue 4.
Derived from the ISO POSIX-2 standard.
NAME
write, writev — write on a file

SYNOPSIS
#include <unistd.h>
ssize_t write(int fildes, const void *buf, size_t nbyte);

#include <sys/uio.h>
ssize_t writev(int fildes, const struct iovec *iov, int iovcnt);

DESCRIPTION
The write() function attempts to write nbyte bytes from the buffer pointed to by buf to the file
associated with the open file descriptor, fildes.

If nbyte is 0, write() will return 0 and have no other results if the file is a regular file; otherwise,
the results are unspecified.

On a regular file or other file capable of seeking, the actual writing of data proceeds from the
position in the file indicated by the file offset associated with fildes. Before successful return
from write(), the file offset is incremented by the number of bytes actually written. On a regular
file, if this incremented file offset is greater than the length of the file, the length of the file will
be set to this file offset.

EX If the O_SYNC flag of the file status flags is set and fildes refers to a regular file, a successful
write() does not return until the data is delivered to the underlying hardware.

On a file not capable of seeking, writing always takes place starting at the current position. The
value of a file offset associated with such a device is undefined.

If the O_APPEND flag of the file status flags is set, the file offset will be set to the end of the file
prior to each write and no intervening file modification operation will occur between changing
the file offset and the write operation.

EX If a write() requests that more bytes be written than there is room for (for example, the ulimit
or the physical end of a medium), only as many bytes as there is room for will be written. For
example, suppose there is space for 20 bytes more in a file before reaching a limit. A write of 512
bytes will return 20. The next write of a non-zero number of bytes will give a failure return
(except as noted below) and the implementation will generate a SIGXFSZ signal for the process.

UX If write() is interrupted by a signal before it writes any data, it will return −1 with errno set to
[EINTR].

FIPS If write() is interrupted by a signal after it successfully writes some data, it will return the
number of bytes written.

If the value of nbyte is greater than (SSIZE_MAX), the result is implementation-dependent.

After a write() to a regular file has successfully returned:
• Any successful read() from each byte position in the file that was modified by that write will
return the data specified by the write() for that position until such byte positions are again
modified.
• Any subsequent successful write() to the same byte position in the file will overwrite that file
data.

Write requests to a pipe or FIFO will be handled the same as a regular file with the following
exceptions:
write()

There is no file offset associated with a pipe, hence each write request will append to the end of the pipe.

Write requests of {PIPE_BUF} bytes or less will not be interleaved with data from other processes doing writes on the same pipe. Writes of greater than {PIPE_BUF} bytes may have data interleaved, on arbitrary boundaries, with writes by other processes, whether or not the O_NONBLOCK flag of the file status flags is set.

If the O_NONBLOCK flag is clear, a write request may cause the process to block, but on normal completion it will return nbyte.

If the O_NONBLOCK flag is set, write() requests will be handled differently, in the following ways:

- The write() function will not block the process.

- A write request for {PIPE_BUF} or fewer bytes will have the following effect: If there is sufficient space available in the pipe, write() will transfer all the data and return the number of bytes requested. Otherwise, write() will transfer no data and return −1 with errno set to [EAGAIN].

- A write request for more than {PIPE_BUF} bytes will case one of the following:
  a. When at least one byte can be written, transfer what it can and return the number of bytes written. When all data previously written to the pipe is read, it will transfer at least {PIPE_BUF} bytes.
  b. When no data can be written, transfer no data and return −1 with errno set to [EAGAIN].

When attempting to write to a file descriptor (other than a pipe or FIFO) that supports non-blocking writes and cannot accept the data immediately:

- If the O_NONBLOCK flag is clear, write() will block until the data can be accepted.

- If the O_NONBLOCK flag is set, write() will not block the process. If some data can be written without blocking the process, write() will write what it can and return the number of bytes written. Otherwise, it will return −1 and errno will be set to [EAGAIN].

Upon successful completion, where nbyte is greater than 0, write() will mark for update the st_ctime and st_mtime fields of the file, and if the file is a regular file, the S_ISUID and S_ISGID bits of the file mode may be cleared.

UX

If fildes refers to a STREAM, the operation of write() is determined by the values of the minimum and maximum nbyte range ("packet size") accepted by the STREAM. These values are determined by the topmost STREAM module. If nbyte falls within the packet size range, nbyte bytes will be written. If nbyte does not fall within the range and the minimum packet size value is 0, write() will break the buffer into maximum packet size segments prior to sending the data downstream (the last segment may contain less than the maximum packet size). If nbyte does not fall within the range and the minimum value is non-zero, write() will fail with errno set to [ERANGE]. Writing a zero-length buffer (nbyte is 0) to a STREAMS device sends 0 bytes with 0 returned. However, writing a zero-length buffer to a STREAMS-based pipe or FIFO sends no message and 0 is returned. The process may issue I_SWROPT ioctl() to enable zero-length messages to be sent across the pipe or FIFO.

When writing to a STREAM, data messages are created with a priority band of 0. When writing to a STREAM that is not a pipe or FIFO:

- If O_NONBLOCK is clear, and the STREAM cannot accept data (the STREAM write queue is full due to internal flow control conditions), write() will block until data can be accepted.
If O_NONBLOCK is set and the STREAM cannot accept data, write() will return −1 and set errno to [EAGAIN].

If O_NONBLOCK is set and part of the buffer has been written while a condition in which the STREAM cannot accept additional data occurs, write() will terminate and return the number of bytes written.

In addition, write() and writev() will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of errno does not reflect the result of write() or writev() but reflects the prior error.

The writev() function is equivalent to write(), but gathers the output data from the iovcnt buffers specified by the members of the iov array: iov[0], iov[1], ..., iov[iovcnt -1]. iovcnt is valid if greater than 0 and less than or equal to [IOV_MAX], defined in <limits.h>.

Each iovec entry specifies the base address and length of an area in memory from which data should be written. The writev() function will always write a complete area before proceeding to the next.

If fildes refers to a regular file and all of the iov_len members in the array pointed to by iov are 0, writev() will return 0 and have no other effect. For other file types, the behaviour is unspecified.

If the sum of the iov_len values is greater than SSIZE_MAX, the operation fails and no data is transferred.

RETURN VALUE

Upon successful completion, write() will return the number of bytes actually written to the file associated with fildes. This number will never be greater than nbyte. Otherwise, −1 is returned and errno is set to indicate the error.

Upon successful completion, writev() returns the number of bytes actually written. Otherwise, it returns a value of −1, the file-pointer remains unchanged, and errno is set to indicate an error.

ERRORS

The write() and writev() functions will fail if:

[EAGAIN] The O_NONBLOCK flag is set for the file descriptor and the process would be delayed in the write() operation.

[EBADF] The fildes argument is not a valid file descriptor open for writing.

[EINVAL] An attempt was made to write a file that exceeds the implementation-dependent maximum file size or the process' file size limit.

[EIO] The write operation was terminated due to the receipt of a signal, and no data was transferred.

[EIO] A physical I/O error has occurred.

[ENOSPC] There was no free space remaining on the device containing the file.

[EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by any process, or that only has one end open. A SIGPIPE signal will also be sent to the process.
The write() function will fail if:

- [ERANGE] The transfer request size was outside the range supported by the STREAMS file associated with fildes.

- [EINVAL] The sum of the iov_len values in the iov array would overflow an ssize_t.

The writev() function will fail if:

- [EINVAL] The iovcnt argument was less than or equal to 0, or greater than [IOV_MAX].

The write() and writev() functions may fail if:

- [EINVAL] The STREAM or multiplexer referenced by fildes is linked (directly or indirectly) downstream from a multiplexer.

- [ENXIO] A request was made of a non-existent device, or the request was outside the capabilities of the device.

- [ENXIO] A hangup occurred on the STREAM being written to.

A write to a STREAMS file may fail if an error message has been received at the STREAM head. In this case, errno is set to the value included in the error message.

The writev() function may fail and set errno to:

- [EINVAL] The iovcnt argument was less than or equal to 0, or greater than [IOV_MAX].

SEE ALSO

chmod(), creat(), dup(), fcntl(), getrlimit(), lseek(), open(), pipe(), ulimit(), <limits.h>, <stropts.h>, <sys/uio.h>, <unistd.h>.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- The type of the argument buf is changed from char * to const void*, and the type of the argument nbyte is changed from unsigned to size_t.

- The DESCRIPTION section is changed:
  — to indicate that writing at end-of-file is atomic
  — to identify that [SSIZE_MAX] is now used to determine the maximum value of nbyte
  — to indicate the consequences of activities after a call to the write() function
  — To improve clarity, the text describing operations on pipes or FIFOs when O_NONBLOCK is set is restructured.

Other changes are incorporated as follows:

- The header <unistd.h> is added to the SYNOPSIS section.

- Reference to ulimit in the DESCRIPTION section is marked as an extension.

- Reference to the process’ file size limit and the ulimit() function are marked as extensions in the description of the [EFBIG] error.

- The [ENXIO] error is marked as an extension.

- The APPLICATION USAGE section is removed.

- The description of [EINVAL] is amended.
Issue 4, Version 2
The following changes are incorporated for X/OPEN UNIX conformance:

- The `writev()` function is added to the **SYNOPSIS**.
- The **DESCRIPTION** is updated to describe the reading of data from STREAMS files, an operational description of the `writev()` function is included, and a statement is added indicating that SIGXFSZ will be generated if an attempted write operation would cause the maximum file size to be exceeded.
- The **RETURN VALUE** section is updated to describe values returned by the `writev()` function.
- The **ERRORS** section has been restructured to describe errors that apply to both `write()` and `writev()` apart from those that apply to `writev()` specifically. The [EIO], [ERANGE] and [EINVAL] errors are also added.
NAME
y0, y1, yn — Bessel functions of the second kind

SYNOPSIS
EX
#include <math.h>
double y0(double x);
double y1 (double x);
double yn (int n, double x);

DESCRIPTION
The y0(), y1() and yn() functions compute Bessel functions of x of the second kind of orders 0, 1 and n respectively. The value of x must be positive.

RETURN VALUE
Upon successful completion, y0(), y1() and yn() will return the relevant Bessel value of x of the second kind.

If x is NaN, NaN is returned and errno may be set to [EDOM].

If the x argument to y0(), y1() or yn() is negative, −HUGE_VAL or NaN is returned, and errno may be set to [EDOM].

If x is 0.0, −HUGE_VAL is returned and errno may be set to [ERANGE] or [EDOM].

If the correct result would cause underflow, 0.0 is returned and errno may be set to [ERANGE].

If the correct result would cause overflow, −HUGE_VAL or 0.0 is returned and errno may be set to [ERANGE].

ERRORS
The y0(), y1() and yn() functions may fail if:

[EDOM] The value of x is negative or NaN.

[ERANGE] The value of x is too large in magnitude, or x is 0.0, or the correct result would cause overflow or underflow.

No other errors will occur.

APPLICATION USAGE
An application wishing to check for error situations should set errno to 0 before calling y0(), y1() or yn(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

SEE ALSO
isnan(), j0(), <math.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated in this issue:

• Removed references to matherr().

• The RETURN VALUE and ERRORS sections are substantially rewritten to rationalise error handling in the mathematics functions.
Chapter 4

Headers

This chapter describes the contents of headers used by the X/Open functions, macros and external variables.

Headers contain the definition of symbolic constants, common structures, preprocessor macros and defined types. Each function in Chapter 3 specifies the headers that an application must include in order to use that function. In most cases only one header is required. These headers are present on an application development system; they do not have to be present on the target execution system.
NAME
assert.h — verify program assertion

SYNOPSIS
#include <assert.h>

DESCRIPTION
The <assert.h> header defines the assert() macro. It refers to the macro NDEBUG which is not defined in the header. If NDEBUG is defined as a macro name before the inclusion of this header, the assert() macro is defined simply as:

#define assert(ignore)((void) 0)

otherwise the macro behaves as described in assert().

The assert() macro is implemented as a macro, not as a function. If the macro definition is suppressed in order to access an actual function, the behaviour is undefined.

SEE ALSO
assert().

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.
NAME

cpio.h — cpio archive values

SYNOPSIS

```
#include <cpio.h>
```

DESCRIPTION

Values needed by the `c_mode` field of the `cpio` archive format are described by:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Value (octal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_IRUSR</td>
<td>read by owner</td>
<td>0000400</td>
</tr>
<tr>
<td>C_IWUSR</td>
<td>write by owner</td>
<td>0000200</td>
</tr>
<tr>
<td>C_IXUSR</td>
<td>execute by owner</td>
<td>0000100</td>
</tr>
<tr>
<td>C_IRGRP</td>
<td>read by group</td>
<td>0000040</td>
</tr>
<tr>
<td>C_IWGRP</td>
<td>write by group</td>
<td>0000020</td>
</tr>
<tr>
<td>C_IXGRP</td>
<td>execute by group</td>
<td>0000010</td>
</tr>
<tr>
<td>C_IROTH</td>
<td>read by others</td>
<td>0000004</td>
</tr>
<tr>
<td>C_IWOTH</td>
<td>write by others</td>
<td>0000002</td>
</tr>
<tr>
<td>C_IXOTH</td>
<td>execute by others</td>
<td>0000001</td>
</tr>
<tr>
<td>C_ISUID</td>
<td>set user ID</td>
<td>0004000</td>
</tr>
<tr>
<td>C_ISGID</td>
<td>set group ID</td>
<td>0002000</td>
</tr>
<tr>
<td>C_ISVTX</td>
<td>on directories, restricted deletion flag</td>
<td>0001000</td>
</tr>
<tr>
<td>C_ISDIR</td>
<td>directory</td>
<td>0040000</td>
</tr>
<tr>
<td>C_ISFIFO</td>
<td>FIFO</td>
<td>0010000</td>
</tr>
<tr>
<td>C_ISREG</td>
<td>regular file</td>
<td>0100000</td>
</tr>
<tr>
<td>C_ISBLK</td>
<td>block special</td>
<td>0060000</td>
</tr>
<tr>
<td>C_ISCHR</td>
<td>character special</td>
<td>0020000</td>
</tr>
<tr>
<td>C_ISCTG</td>
<td>reserved</td>
<td>0110000</td>
</tr>
<tr>
<td>C_ISLNK</td>
<td>symbolic link</td>
<td>0120000</td>
</tr>
<tr>
<td>C_ISSOCK</td>
<td>socket</td>
<td>0140000</td>
</tr>
</tbody>
</table>

The header defines the symbolic constant:

```
MAGIC "070707"
```

SEE ALSO

cpio, the XCU specification.

CHANGE HISTORY

First released in Issue 3 of the referenced Headers specification.

Derived from the POSIX.1-1988 standard.

Issue 4, Version 2

Descriptions for C_ISLNK and C_ISSOCK are provided; formerly, these were listed as “Reserved”.

System Interfaces and Headers Issue 4, Version 2 747
NAME
ctype.h — character types

SYNOPSIS
#include <ctype.h>

DESCRIPTION
The <ctype.h> header declares the following as functions and may also define them as macros:

    int isalnum(int c);
    int isalpha(int c);
    int isascii(int c);
    int iscntrl(int c);
    int isdigit(int c);
    int isgraph(int c);
    int islower(int c);
    int isprint(int c);
    int ispunct(int c);
    int isspace(int c);
    int isupper(int c);
    int isxdigit(int c);

EX
    int toascii(int c);
    int tolower(int c);
    int toupper(int c);

    int _toupper(int c);
    int _tolower(int c);

The following are defined as macros:

EX
    int _toupper(int c);
    int _tolower(int c);

SEE ALSO
    isalnum(), isalpha(), isascii(), iscntrl(), isdigit(), isgraph(), islower(), isprint(), ispunct(), isspace(),
    isupper(), isxdigit(), mblen(), mbstowcs(), mbtowc(), setlocale(), toascii(), tolower(), _tolower(),
    toupper(), _toupper(), wcstombs(), wctomb(), <locale.h>.

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:
   • The function declarations in this header are expanded to full ISO C prototypes.
NAME
dirent.h — format of directory entries

SYNOPSIS
#include <dirent.h>

DESCRIPTION
The internal format of directories is unspecified.

The <dirent.h> header defines the following data type through typedef:

```c
DIR A type representing a directory stream.
```

It also defines the structure `dirent` which includes the following members:

```c
struct dirent {
    ino_t   d_ino; /* file serial number */
    char d_name[ ]; /* name of entry */
};
```

The type `ino_t` is defined as described in `<sys/types.h>`.

The character array `d_name` is of unspecified size, but the number of bytes preceding the terminating null byte will not exceed `NAME_MAX`.

The following are declared as functions and may also be defined as macros:

```c
int closedir(DIR * dirp);
DIR * opendir(const char * dirname);
struct dirent * readdir(DIR * dirp);
void rewinddir(DIR * dirp);
void seekdir(DIR * dirp, long int loc);
long int telldir(DIR * dirp);
```

SEE ALSO
`closedir()`, `opendir()`, `readdir()`, `rewinddir()`, `seekdir()`, `telldir()`, `<sys/types.h>`.

CHANGE HISTORY
First released in Issue 2.

Issue 4

The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- The function declarations in this header are expanded to full ISO C prototypes.
- A statement is added to the DESCRIPTION section indicating that the internal format of directories is unspecified. Also in the description of the `d_name` field, the text is changed to indicate “bytes” rather than (possibly multi-byte) “characters”.

Another change is incorporated as follows:

- Reference to type `ino_t` is marked as an extension, as are references to the `seekdir()` and `telldir()` functions.
NAME
errno.h — system error numbers

SYNOPSIS
#include <errno.h>

DESCRIPTION
The `<errno.h>` header provides a declaration for `errno` and gives non-zero values for the
following symbolic constants. Their values are unique except as noted below:

UX E2BIG Argument list too long.
UX EACCESS Permission denied.
UX EADDRINUSE Address in use.
UX EADDRNOTAVAIL Address not available.
UX EAFNOSUPPORT Address family not supported.
UX EAGAIN Resource unavailable, try again (may be the same value as EWOULDBLOCK).
UX EALREADY Connection already in progress.
UX EBADF Bad file descriptor.
UX EBADMSG Bad message.
UX EBUSY Device or resource busy.
UX ECHILD No child processes.
UX ECONNABORTED Connection aborted.
UX ECONNREFUSED Connection refused.
UX ECONNRESET Connection reset.
UX EDENALD Resource deadlock would occur.
UX EDESTADDRREQ Destination address required.
UX EDOM Mathematics argument out of domain of function.
UX EDQUOT Reserved.
UX EEXIST File exists.
UX EINVAL Invalid argument.
UX EIO I/O error.
UX EISCONN Socket is connected.
UX EISDIR Is a directory.
UX ELOOP Too many levels of symbolic links.
UX EMFILE Too many open files.
UX EMLINK Too many links.
UX EMSGSIZE Message too large.
UX EMLIHHOP Reserved.
UX ENAMETOOLONG Filename too long.
UX ENETDOWN Network is down.
UX ENETUINREACH Network unreachable.
UX ENFILE Too many files open in system.
UX ENOBFS No buffer space available.
UX ENODATA No message is available on the STREAM head read queue.
UX ENODEV No such device.
UX ENOENT No such file or directory.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENOEXEC</td>
<td>Executable file format error.</td>
</tr>
<tr>
<td>ENOLCK</td>
<td>No locks available.</td>
</tr>
<tr>
<td>ENOLINK</td>
<td>Reserved.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>Not enough space.</td>
</tr>
<tr>
<td>ENOMSG</td>
<td>No message of the desired type.</td>
</tr>
<tr>
<td>ENOPROTOOPT</td>
<td>Protocol not available.</td>
</tr>
<tr>
<td>ENOSP</td>
<td>No space left on device.</td>
</tr>
<tr>
<td>ENOSR</td>
<td>No STREAM resources.</td>
</tr>
<tr>
<td>ENOSTR</td>
<td>Not a STREAM.</td>
</tr>
<tr>
<td>ENOSYS</td>
<td>Function not supported.</td>
</tr>
<tr>
<td>ENOTCONN</td>
<td>The socket is not connected.</td>
</tr>
<tr>
<td>ENOTDIR</td>
<td>Not a directory.</td>
</tr>
<tr>
<td>ENOTEMPTY</td>
<td>Directory not empty.</td>
</tr>
<tr>
<td>ENOTSOCK</td>
<td>Not a socket.</td>
</tr>
<tr>
<td>ENOTTY</td>
<td>Inappropriate I/O control operation.</td>
</tr>
<tr>
<td>EXNIO</td>
<td>No such device or address.</td>
</tr>
<tr>
<td>EOPNOTSUPP</td>
<td>Operation not supported on socket.</td>
</tr>
<tr>
<td>EOVERFLOW</td>
<td>Value too large to be stored in data type.</td>
</tr>
<tr>
<td>FIPS EPERM</td>
<td>Operation not permitted.</td>
</tr>
<tr>
<td>EPIPE</td>
<td>Broken pipe.</td>
</tr>
<tr>
<td>EPROTO</td>
<td>Protocol error.</td>
</tr>
<tr>
<td>EPROTONOSUPPORT</td>
<td>Protocol not supported.</td>
</tr>
<tr>
<td>EPROTOTYPE</td>
<td>Socket type not supported.</td>
</tr>
<tr>
<td>ERANGE</td>
<td>Result too large.</td>
</tr>
<tr>
<td>EROFS</td>
<td>Read-only file system.</td>
</tr>
<tr>
<td>ESPIPE</td>
<td>Invalid pipe.</td>
</tr>
<tr>
<td>ESRC</td>
<td>No such process.</td>
</tr>
<tr>
<td>ESTALE</td>
<td>Reserved.</td>
</tr>
<tr>
<td>ETIME</td>
<td>Stream ioctl() timeout.</td>
</tr>
<tr>
<td>ETIMEDOUT</td>
<td>Connection timed out.</td>
</tr>
<tr>
<td>EX EWOULDBLOCK</td>
<td>Operation would block (may be the same value as EAGAIN).</td>
</tr>
<tr>
<td>EXDEV</td>
<td>Cross-device link.</td>
</tr>
</tbody>
</table>

**APPLICATION USAGE**

Additional error numbers may be defined on XSI-conformant systems. See Section 2.3.1 on page 31.

**SEE ALSO**

Section 2.3 on page 25.

**CHANGE HISTORY**

First released in Issue 1.

Derived from Issue 1 of the SVID.

**Issue 4**

The following changes are incorporated in this issue:

- The [EILSEQ] error is added and marked as an EX interface.
- The [ENOTBLK] error is withdrawn.

**Issue 4, Version 2**

The EADDRINUSE, EADDRNOTAVAIL, EAFNOSUPPORT, EALREADY, EBADMSG, ECONNABORTED, ECONNREFUSED, ECONNRESET, EDESTADDRREQ, EDQUOT,
EHOSTUNREACH, EINPROGRESS, EISCONN, ELOOP, EMSGSIZE, EMULTIHOP, ENETDOWN, ENETUNREACH, ENOBUFS, ENODATA, ENOLINK, ENOPROTOOPT, ENOSR, ENOSTR, ENOTCONN, ENOTSOCK, EOPNOTSUPP, EOVERFLOW, EPROTO, EPROTONOSUPPORT, EPROTOTYPE, ESTALE, ETIME, ETIMEDOUT and EWOULDBLOCK errors are added in the UX context.
NAME
fcntl.h — file control options

SYNOPSIS
#include <fcntl.h>

DESCRIPTION
The <fcntl.h> header defines the following requests and arguments for use by the functions
fcntl() and open().

Values for cmd used by fcntl() (the following values are unique):
- F_DUPFD   Duplicate file descriptor.
- F_GETFD   Get file descriptor flags.
- F_SETFD   Set file descriptor flags.
- F_GETFL   Get file status flags and file access modes.
- F_SETFL   Set file status flags.
- F_GETLK   Get record locking information.
- F_SETLK   Set record locking information.
- F_SETLKW  Set record locking information; wait if blocked.

File descriptor flags used for fcntl():
- FD_CLOEXEC Close the file descriptor upon execution of an exec family function.

Values for l_type used for record locking with fcntl() (the following values are unique):
- F_RDLCK Shared or read lock.
- F_UNLCK Unlock.
- F_WRLCK Exclusive or write lock.

The values used for l_whence, SEEK_SET, SEEK_CUR and SEEK_END are defined as described in <unistd.h>.

The following four sets of values for oflag used by open() are bitwise distinct:
- O_CREAT Create file if it does not exist.
- O_EXCL  Exclusive use flag.
- O_NOCTTY Do not assign controlling terminal.
- O_TRUNC Truncate flag.

File status flags used for open() and fcntl():
- O_APPEND Set append mode.
- O_NONBLOCK Non-blocking mode.

Mask for use with file access modes:
- O_ACCMODE Mask for file access modes.

File access modes used for open() and fcntl():
- O_RDONLY Open for reading only.
- O_RDWR Open for reading and writing.
- O_WRONLY Open for writing only.

The symbolic names for file modes for use as values of mode_t are defined as described in <sys/types.h>. 
The structure `flock` describes a file lock. It includes the following members:

- `short l_type`: type of lock; F_RDLCK, F_WRLCK, F_UNLCK
- `short l_whence`: flag for starting offset
- `off_t l_start`: relative offset in bytes
- `off_t l_len`: size; if 0 then until EOF
- `pid_t l_pid`: process ID of the process holding the lock; returned with F_GETLK

The `mode_t`, `off_t` and `pid_t` types are defined as described in `<sys/types.h>`.

The following are declared as functions and may also be defined as macros:

```c
int creat(const char *path, mode_t mode);
int fcntl(int fildes, int cmd, ...);
int open(const char *path, int oflag, ...);
```

Inclusion of the `<fcntl.h>` header may also make visible all symbols from `<sys/stat.h>` and `<unistd.h>`.

**SEE ALSO**
- `creat()`, `exec`, `fcntl()`, `open()`, `<sys/stat.h>`, `<sys/types.h>`, `<unistd.h>`.

**CHANGE HISTORY**
First released in Issue 1.

Derived from Issue 1 of the SVID.

**Issue 4**
The following change is incorporated for alignment with the ISO POSIX-1 standard:
- The function declarations in this header are expanded to full ISO C prototypes.

Other changes are incorporated as follows:
- A reference to `<unistd.h>` is added for the definition of `l_whence`, SEEK_SET, SEEK_CUR and SEEK_END, and marked as an extension.
- A reference to `<sys/stat.h>` is added for the symbolic names of file modes used as values of `mode_t`, and marked as an extension.
- A reference to `<sys/types.h>` is added for the definition of `mode_t`, `off_t` and `pid_t`, and marked as an extension.
- A warning is added indicating that inclusion of `<fcntl.h>` may also make visible all symbols from `<sys/stat.h>` and `<unistd.h>`. This is marked as an extension.
NAME
float.h — floating types

SYNOPSIS
#include <float.h>

DESCRIPTION
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.

The following parameters are used to define the model for each floating-point type:

- \( s \)  sign (±1)
- \( b \)  base or radix of exponent representation (an integer > 1)
- \( e \)  exponent (an integer between a minimum \( e_{\text{min}} \) and a maximum \( e_{\text{max}} \))
- \( p \)  precision (the number of base-\( b \) digits in the significand)
- \( f_k \)  non-negative integers less than \( b \) (the significand digits)

A normalised floating-point number \( x \) (\( f_1 > 0 \) if \( x \neq 0 \)) is defined by the following model:

\[
x = s \times b^e \times \sum_{k=1}^{p} f_k \times b^{-k}, \quad e_{\text{min}} \leq e \leq e_{\text{max}}
\]

FLT_RADIX will be a constant expression suitable for use in the \#if preprocessing directives. All
except FLT_RADIX and FLT_ROUNDS have separate names for all three floating-point types.
The floating-point model representation is provided for all macro names except FLT_ROUNDS.

The rounding mode for floating-point addition is characterised by the value of FLT_ROUNDS:

- \(-1\)  indeterminable
- \(0\)  toward 0.0
- \(1\)  to nearest
- \(2\)  toward positive infinity
- \(3\)  toward negative infinity

All other values for FLT_ROUNDS characterise implementation-dependent rounding behaviour.
The macro names given in the following list will be defined as expressions with values that are
equal or greater in magnitude (absolute value) to those shown, with the same sign.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLT_RADIX</td>
<td>radix of exponent representation, $b$</td>
<td>2</td>
</tr>
<tr>
<td>FLT_MANT_DIG</td>
<td>number of base-FLT_RADIX digits in the floating-point significand, $p$</td>
<td></td>
</tr>
<tr>
<td>DBL_MANT_DIG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDBL_MANT_DIG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLT_DIG</td>
<td>number of decimal digits, $q$, such that any floating-point number with $q$</td>
<td>6</td>
</tr>
<tr>
<td>DBL_DIG</td>
<td>number with $q$ decimal digits can be rounded into a floating-point number</td>
<td>10</td>
</tr>
<tr>
<td>LDBL_DIG</td>
<td>number with $p$ radix $b$ digits and back again without change to the $q$</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>decimal digits,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\left\lfloor (p-1) \times \log_{10} b \right\rfloor + \begin{cases} 1 &amp; \text{if } b \text{ is a power of } 10 \ 0 &amp; \text{otherwise} \end{cases}$</td>
<td></td>
</tr>
<tr>
<td>FLT_MIN_EXP</td>
<td>minimum negative integer such that FLT_RADIX raised to that power minus 1 is</td>
<td>$-37$</td>
</tr>
<tr>
<td>DBL_MIN_EXP</td>
<td>a normalised floating-point number, $\epsilon_{\min}$</td>
<td>$-37$</td>
</tr>
<tr>
<td>LDBL_MIN_EXP</td>
<td></td>
<td>$-37$</td>
</tr>
<tr>
<td>FLT_MIN_10_EXP</td>
<td>minimum negative integer such that $10$ raised to that power is in the</td>
<td>$-37$</td>
</tr>
<tr>
<td>DBL_MIN_10_EXP</td>
<td>range of normalised floating point numbers, $\epsilon_{\min}$</td>
<td>$-37$</td>
</tr>
<tr>
<td>LDBL_MIN_10_EXP</td>
<td></td>
<td>$-37$</td>
</tr>
<tr>
<td>FLT_MAX_EXP</td>
<td>maximum integer such that FLT_RADIX raised to that power minus 1 is a</td>
<td></td>
</tr>
<tr>
<td>DBL_MAX_EXP</td>
<td>representable finite floating-point number, $\epsilon_{\max}$</td>
<td></td>
</tr>
<tr>
<td>LDBL_MAX_EXP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLT_MAX_10_EXP</td>
<td>maximum integer such that $10$ raised to that power is in the range of</td>
<td>$37$</td>
</tr>
<tr>
<td>DBL_MAX_10_EXP</td>
<td>representable finite floating-point numbers, $\epsilon_{\max}$</td>
<td>$37$</td>
</tr>
<tr>
<td>LDBL_MAX_10_EXP</td>
<td></td>
<td>$37$</td>
</tr>
</tbody>
</table>

The macro names given in the following list will be defined as expressions with values that will be equal to or greater than those shown.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLT_MAX</td>
<td>maximum representable finite floating-point number, $1E+37$</td>
<td>$1E+37$</td>
</tr>
<tr>
<td>DBL_MAX</td>
<td>$(1 - b^{-p}) \times b^{\epsilon_{\max}}$</td>
<td>$1E+37$</td>
</tr>
<tr>
<td>LDBL_MAX</td>
<td></td>
<td>$1E+37$</td>
</tr>
</tbody>
</table>

The macro names given in the following list will be defined as expressions with values that will be equal to or less than those shown.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLT_EPSILON</td>
<td>the difference between 1.0 and the least value greater that $b^{(1-p)}$</td>
<td>$1E-5$</td>
</tr>
<tr>
<td>DBL_EPSILON</td>
<td>1.0 that is representable in the given floating-point type, $b^{(1-p)}$</td>
<td>$1E-9$</td>
</tr>
<tr>
<td>LDBL_EPSILON</td>
<td></td>
<td>$1E-9$</td>
</tr>
<tr>
<td>FLT_MIN</td>
<td>minimum normalised positive floating-point number, $b^{(\epsilon_{\min}-1)}$</td>
<td>$1E-37$</td>
</tr>
<tr>
<td>DBL_MIN</td>
<td></td>
<td>$1E-37$</td>
</tr>
<tr>
<td>LDBL_MIN</td>
<td></td>
<td>$1E-37$</td>
</tr>
</tbody>
</table>

**CHANGE HISTORY**

First released in Issue 4.

Derived from the ISO C standard.
NAME
fmtmsg.h — message display structures

SYNOPSIS
UX
#include <fmtmsg.h>

DESCRIPTION
The <fmtmsg.h> header defines the following macros, which expand to constant integral expressions:

- MM_HARD Source of the condition is hardware.
- MM_SOFT Source of the condition is software.
- MM_FIRM Source of the condition is firmware.
- MM_APPL Condition detected by application.
- MM_UTIL Condition detected by utility.
- MM_OPSYS Condition detected by operating system.
- MM_RECOVER Recoverable error.
- MM_NRECOV Non-recoverable error.
- MM_HALT Error causing application to halt.
- MM_ERROR Application has encountered a non-fatal fault.
- MM_WARNING Application has detected unusual non-error condition.
- MM_INFO Informative message.
- MM_NOSEV No severity level provided for the message.
- MM_PRINT Display message on standard error.
- MM_CONSOLE Display message on system console.

The table below indicates the null values and identifiers for fmtmsg() arguments. The <fmtmsg.h> header defines the macros in the Identifier column, which expand to constant expressions that expand to expressions of the type indicated in the Type column:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Null-Value</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>char*</td>
<td>(char*)0</td>
<td>MM_NULLLBL</td>
</tr>
<tr>
<td>severity</td>
<td>int</td>
<td>0</td>
<td>MM_NULLSEV</td>
</tr>
<tr>
<td>class</td>
<td>long int</td>
<td>0L</td>
<td>MM_NULLMC</td>
</tr>
<tr>
<td>text</td>
<td>char*</td>
<td>(char*)0</td>
<td>MM_NULLTXT</td>
</tr>
<tr>
<td>action</td>
<td>char*</td>
<td>(char*)0</td>
<td>MM_NULLACT</td>
</tr>
<tr>
<td>tag</td>
<td>char*</td>
<td>(char*)0</td>
<td>MM_NULLTAG</td>
</tr>
</tbody>
</table>

The <fmtmsg.h> header also defines the following macros for use as return values for fmtmsg():

- MM_OK The function succeeded.
- MM_NOTOK The function failed completely.
- MM_NOMSG The function was unable to generate a message on standard error, but otherwise succeeded.
- MM_NOCON The function was unable to generate a console message, but otherwise succeeded.

The following is declared as a function and may also be defined as a macro:

```
int fmtmsg(long classification, const char *label, int severity,
           const char *text, const char *action, const char *tag);
```
SEE ALSO
   fmtmsg().

CHANGE HISTORY
   First released in Issue 4, Version 2.
NAME

fnmatch.h — filename-matching types

SYNOPSIS

#include <fnmatch.h>

DESCRIPTION

The <fnmatch.h> header defines the flags and return value used by the fnmatch() function. The following constants are defined:

- FNM_NOMATCH: The string does not match the specified pattern.
- FNM_PATHNAME: Slash in string only matches slash in pattern.
- FNM_PERIOD: Leading period in string must be exactly matched by period in pattern.
- FNM_NOESCAPE: Disable backslash escaping.
- FNM_NOSYS: The implementation does not support this function.

The following is declared as a function and may also be declared as a macro:

```c
int fnmatch(const char *pattern, const char *string, int flags);
```

SEE ALSO

fnmatch(), the XCU specification.

CHANGE HISTORY

First released in Issue 4.

Derived from the ISO POSIX-2 standard.
NAME

ftw.h — file tree traversal

SYNOPSIS

```c
#include <ftw.h>
```

DESCRIPTION

The `<ftw.h>` header defines the FTW structure that includes at least the following members:

```c
typedef struct FTW {
    int base;
    int level;
} FTW;
```

The `<ftw.h>` header defines macros for use as values of the third argument to the application-supplied function that is passed as the second argument to `ftw()` and `nftw()`:

- `FTW_F` File.
- `FTW_D` Directory.
- `FTW_DNR` Directory without read permission.
- `FTW_NS` Unknown type, `stat()` failed.
- `FTW_SL` Symbolic link.
- `FTW_SLN` Symbolic link that names a non-existent file.

The `<ftw.h>` header defines macros for use as values of the fourth argument to `nftw()`:

- `FTW_PHYS` Physical walk, does not follow symbolic links. Otherwise, `nftw()` will follow links but will not walk down any path that crosses itself.
- `FTW_MOUNT` The walk will not cross a mount point.
- `FTW_DEPTH` All subdirectories will be visited before the directory itself.
- `FTW_CHDIR` The walk will change to each directory before reading it.

The following are declared as functions and may also be defined as macros:

```c
int ftw(const char *path,
        int (*fn)(const char *, const struct stat *, int), int ndirs);
```

```c
int nftw(const char *path, int (*fn)(
        const char *, const struct stat *, int, struct FTW*),
        int depth, int flags);
```

The `<ftw.h>` header defines the `stat` structure and the symbolic names for `st_mode` and the file type test macros as described in `<sys/stat.h>`.

Inclusion of the `<ftw.h>` header may also make visible all symbols from `<sys/stat.h>`.

SEE ALSO

`ftw()`, `nftw()`, `<sys/stat.h>`.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated in this issue:

- The function declarations in this header are expanded to full ISO C prototypes.
- A reference to `<sys/stat.h>` is added for the definition of the `stat` structure, the symbolic names for `st_mode` and the file type test macros.
A warning is added indicating that inclusion of `<ftw.h>` may also make visible all symbols from `<sys/stat.h>`.

**Issue 4, Version 2**

The following changes are incorporated in the *DESCRIPTION* for X/OPEN UNIX conformance:

- The **FTW** structure is defined.
- The **nftw()** function is declared by the header and is mentioned as one of the functions to which the first list of macros applies.
- **FTW_SL** and **FTW_SLN** are added to the first list of macros to handle symbolic links.
- Macros for use as values of the fourth argument to **nftw()** are defined.
NAME

glob.h — pathname pattern-matching types

SYNOPSIS

#include <glob.h>

DESCRIPTION

The <glob.h> header defines the structures and symbolic constants used by the glob() function.

The structure type glob_t contains at least the following members:

size_t gl_pathc count of paths matched by pattern
char **gl_pathv pointer to a list of matched pathnames
size_t gl_offs slots to reserve at the beginning of gl_pathv

The following constants are provided as values for the flags argument:

GLOB_APPEND Append generated pathnames to those previously obtained.
GLOB_DOOFFS Specify how many null pointers to add to the beginning of pglob->gl_pathv.
GLOB_ERR Cause glob() to return on error.
GLOB_MARK Each pathname that is a directory that matches pattern has a slash appended.
GLOB_NOCHECK If pattern does not match any pathname, then return a list consisting of only pattern.
GLOB_NOESCAPE Disable backslash escaping.
GLOB_NOSORT Do not sort the pathnames returned.

The following constants are defined as error return values:

GLOB_ABORTED The scan was stopped because GLOB_ERR was set or (*errfunc)() returned non-zero.
GLOB_NOMATCH The pattern does not match any existing pathname, and GLOB_NOCHECK was not set in flags.
GLOB_NOSPACE An attempt to allocate memory failed.
GLOB_NOSYS The implementation does not support this function.

The following are declared as functions and may also be declared as macros:

int glob(const char *pattern, int flags,
        int (*errfunc)(const char *epath, int errno), glob_t *pglob);
void globfree (glob_t *pglob);

The implementation may define additional macros or constants using names beginning with GLOB_.

SEE ALSO

glob(), the XCU specification.

CHANGE HISTORY

First released in Issue 4.

Derived from the ISO POSIX-2 standard.
NAME
grp.h — group structure

SYNOPSIS
#include <grp.h>

DESCRIPTION
The <grp.h> header declares the structure group which includes the following members:

- char *gr_name: the name of the group
- gid_t gr_gid: numerical group ID
- char **gr_mem: pointer to a null-terminated array of character pointers to member names.

The gid_t type is defined as described in <sys/types.h>.

The following are declared as functions and may also be defined as macros:

- struct group *getgrgid(gid_t gid);
- struct group *getgrnam(const char *name);
- struct group *getgrent(void);
- void endgrent(void);
- void setgrent(void);

SEE ALSO
endgrent(), getgrgid(), getgrnam(), <sys/types.h>.

CHANGE HISTORY
First released in Issue 1.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:
- The function declarations in this header are expanded to full ISO C prototypes.

Another change is incorporated as follows:
- A reference to <sys/types.h> is added for the definition of gid_t and marked as an extension.

Issue 4, Version 2
For X/OPEN UNIX conformance, the getgrent(), endgrent() and setgrent() functions are added to the list of functions declared in this header.
NAME
iconv.h — codeset conversion facility

SYNOPSIS
EX
#include <iconv.h>

DESCRIPTION
The <iconv.h> header defines the following data type through typedef:

iconv_t Identifies the conversion from one codeset to another.

The following are declared as functions and may also be declared as macros:

iconv_t iconv_open(const char *tocode, const char *fromcode);
size_t iconv(iconv_t cd, char **inbuf, size_t *inbytesleft,
char **outbuf, size_t *outbytesleft);
int iconv_close(iconv_t cd);

SEE ALSO
iconv_open(), iconv(), iconv_close().

CHANGE HISTORY
First released in Issue 4.
NAME
langinfo.h — language information constants

SYNOPSIS
#include <langinfo.h>

DESCRIPTION
The <langinfo.h> header contains the constants used to identify items of langinfo data (see nl_langinfo()). The type of the constants, nl_item, is defined as described in <nl_types.h>. The following constants are defined on all XSI-conformant systems.

The entries under Category indicate in which setlocale() category each item is defined.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Category</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODESET</td>
<td>LC_CTYPE</td>
<td>codeset name</td>
</tr>
<tr>
<td>D_T_FMT</td>
<td>LC_TIME</td>
<td>string for formatting date and time</td>
</tr>
<tr>
<td>D_FMT</td>
<td>LC_TIME</td>
<td>date format string</td>
</tr>
<tr>
<td>T_FMT</td>
<td>LC_TIME</td>
<td>time format string</td>
</tr>
<tr>
<td>T_FMT_AMPM</td>
<td>LC_TIME</td>
<td>a.m. or p.m. time format string</td>
</tr>
<tr>
<td>AM_STR</td>
<td>LC_TIME</td>
<td>Ante Meridian affix</td>
</tr>
<tr>
<td>PM_STR</td>
<td>LC_TIME</td>
<td>Post Meridian affix</td>
</tr>
<tr>
<td>DAY_1</td>
<td>LC_TIME</td>
<td>name of the first day of the week (for example, Sunday)</td>
</tr>
<tr>
<td>DAY_2</td>
<td>LC_TIME</td>
<td>name of the second day of the week (for example, Monday)</td>
</tr>
<tr>
<td>DAY_3</td>
<td>LC_TIME</td>
<td>name of the third day of the week (for example, Tuesday)</td>
</tr>
<tr>
<td>DAY_4</td>
<td>LC_TIME</td>
<td>name of the fourth day of the week (for example, Wednesday)</td>
</tr>
<tr>
<td>DAY_5</td>
<td>LC_TIME</td>
<td>name of the fifth day of the week (for example, Thursday)</td>
</tr>
<tr>
<td>DAY_6</td>
<td>LC_TIME</td>
<td>name of the sixth day of the week (for example, Friday)</td>
</tr>
<tr>
<td>DAY_7</td>
<td>LC_TIME</td>
<td>name of the seventh day of the week (for example, Saturday)</td>
</tr>
<tr>
<td>ABDAY_1</td>
<td>LC_TIME</td>
<td>abbreviated name of the first day of the week</td>
</tr>
<tr>
<td>ABDAY_2</td>
<td>LC_TIME</td>
<td>abbreviated name of the second day of the week</td>
</tr>
<tr>
<td>ABDAY_3</td>
<td>LC_TIME</td>
<td>abbreviated name of the third day of the week</td>
</tr>
<tr>
<td>ABDAY_4</td>
<td>LC_TIME</td>
<td>abbreviated name of the fourth day of the week</td>
</tr>
<tr>
<td>ABDAY_5</td>
<td>LC_TIME</td>
<td>abbreviated name of the fifth day of the week</td>
</tr>
<tr>
<td>ABDAY_6</td>
<td>LC_TIME</td>
<td>abbreviated name of the sixth day of the week</td>
</tr>
<tr>
<td>ABDAY_7</td>
<td>LC_TIME</td>
<td>abbreviated name of the seventh day of the week</td>
</tr>
<tr>
<td>MON_1</td>
<td>LC_TIME</td>
<td>name of the first month of the year</td>
</tr>
<tr>
<td>MON_2</td>
<td>LC_TIME</td>
<td>name of the second month</td>
</tr>
<tr>
<td>MON_3</td>
<td>LC_TIME</td>
<td>name of the third month</td>
</tr>
<tr>
<td>MON_4</td>
<td>LC_TIME</td>
<td>name of the fourth month</td>
</tr>
<tr>
<td>MON_5</td>
<td>LC_TIME</td>
<td>name of the fifth month</td>
</tr>
<tr>
<td>MON_6</td>
<td>LC_TIME</td>
<td>name of the sixth month</td>
</tr>
<tr>
<td>MON_7</td>
<td>LC_TIME</td>
<td>name of the seventh month</td>
</tr>
<tr>
<td>MON_8</td>
<td>LC_TIME</td>
<td>name of the eighth month</td>
</tr>
<tr>
<td>MON_9</td>
<td>LC_TIME</td>
<td>name of the ninth month</td>
</tr>
<tr>
<td>MON_10</td>
<td>LC_TIME</td>
<td>name of the tenth month</td>
</tr>
<tr>
<td>MON_11</td>
<td>LC_TIME</td>
<td>name of the eleventh month</td>
</tr>
<tr>
<td>MON_12</td>
<td>LC_TIME</td>
<td>name of the twelfth month</td>
</tr>
<tr>
<td>ABMON_1</td>
<td>LC_TIME</td>
<td>abbreviated name of the first month</td>
</tr>
</tbody>
</table>
<langinfo.h>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Category</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABMON_2</td>
<td>LC_TIME</td>
<td>abbreviated name of the second month</td>
</tr>
<tr>
<td>ABMON_3</td>
<td>LC_TIME</td>
<td>abbreviated name of the third month</td>
</tr>
<tr>
<td>ABMON_4</td>
<td>LC_TIME</td>
<td>abbreviated name of the fourth month</td>
</tr>
<tr>
<td>ABMON_5</td>
<td>LC_TIME</td>
<td>abbreviated name of the fifth month</td>
</tr>
<tr>
<td>ABMON_6</td>
<td>LC_TIME</td>
<td>abbreviated name of the sixth month</td>
</tr>
<tr>
<td>ABMON_7</td>
<td>LC_TIME</td>
<td>abbreviated name of the seventh month</td>
</tr>
<tr>
<td>ABMON_8</td>
<td>LC_TIME</td>
<td>abbreviated name of the eighth month</td>
</tr>
<tr>
<td>ABMON_9</td>
<td>LC_TIME</td>
<td>abbreviated name of the ninth month</td>
</tr>
<tr>
<td>ABMON_10</td>
<td>LC_TIME</td>
<td>abbreviated name of the tenth month</td>
</tr>
<tr>
<td>ABMON_11</td>
<td>LC_TIME</td>
<td>abbreviated name of the eleventh month</td>
</tr>
<tr>
<td>ABMON_12</td>
<td>LC_TIME</td>
<td>abbreviated name of the twelfth month</td>
</tr>
<tr>
<td>ERA</td>
<td>LC_TIME</td>
<td>era description segments</td>
</tr>
<tr>
<td>ERA_D_FMT</td>
<td>LC_TIME</td>
<td>era date format string</td>
</tr>
<tr>
<td>ERA_D_T_FMT</td>
<td>LC_TIME</td>
<td>era date and time format string</td>
</tr>
<tr>
<td>ERA_T_FMT</td>
<td>LC_TIME</td>
<td>era time format string</td>
</tr>
<tr>
<td>ALT_DIGITS</td>
<td>LC_TIME</td>
<td>alternative symbols for digits</td>
</tr>
<tr>
<td>RADIXCHAR</td>
<td>LC_NUMERIC</td>
<td>radix character</td>
</tr>
<tr>
<td>THOUSEP</td>
<td>LC_NUMERIC</td>
<td>separator for thousands</td>
</tr>
<tr>
<td>YESEXPR</td>
<td>LC_MESSAGES</td>
<td>affirmative response expression</td>
</tr>
<tr>
<td>NOEXPR</td>
<td>LC_MESSAGES</td>
<td>negative response expression</td>
</tr>
<tr>
<td>YESSTR</td>
<td>LC_MESSAGES</td>
<td>affirmative response for yes/no queries</td>
</tr>
<tr>
<td>NOSTR</td>
<td>LC_MESSAGES</td>
<td>negative response for yes/no queries</td>
</tr>
<tr>
<td>CRNCYSTR</td>
<td>LC_MONETARY</td>
<td>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</td>
</tr>
</tbody>
</table>

If the locale's value for `p_cs_precedes` and `n_cs_precedes` do not match, the value of `nl_langinfo(CRNCYSTR)` is unspecified.

The `<langinfo.h>` header declares the following as a function:

```c
char *nl_langinfo(nl_item item);
```

Inclusion of the `<langinfo.h>` header may also make visible all symbols from `<nl_types.h>`.

**APPLICATION USAGE**

Wherever possible, users are advised to use functions compatible with those in the ISO C standard to access items of `langinfo` data. In particular, the `strftime()` function should be used to access date and time information defined in category `LC_TIME`. The `localeconv()` function should be used to access information corresponding to `RADIXCHAR`, `THOUSEP` and `CRNCYSTR`.

**SEE ALSO**

`nl_langinfo()`, `localeconv()`, `strfmon()`, `strftime()`, the XBD specification, Chapter 5, Locale.

**CHANGE HISTORY**

First released in Issue 2.
Issue 4

The following changes are incorporated in this issue:

- The function declarations in this header are expanded to full ISO C prototypes.
- The constants CODESET, T_FMT_AMPM, ERA, ERA_D_FMT, ALT_DIGITS, YESEXPR and NOEXPR are added.
- The constants YESSTR and NOSTR are marked TO BE WITHDRAWN.
- Reference to the Gregorian calendar is removed.
- Constants YESSTR and NOSTR are now defined as belonging to category LC_MESSAGES. Previously they were defined as constants in category LC_ALL.
- A warning is added indicating that inclusion of `<langinfo.h>` may also make visible all symbols from `<nl_types.h>`.
- The APPLICATION USAGE section is expanded to recommend use of the `localeconv()` function.
NAME

libgen.h — definitions for pattern matching functions

SYNOPSIS

#include <libgen.h>

DESCRIPTION

The header <libgen.h> declares the following external variable:

extern char* __loc1  (TO BE WITHDRAWN)
   Used by regex() to report pattern location

The following are declared as functions and may also be defined as macros:

char  *regcmp(const char *string1, ... );
char  *basename (const char *path);
char  *dirname (const char *path);
char  *regex(const char *re1 , const char *subject, ... );

SEE ALSO

regcmp().

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
limits.h — implementation-dependent constants

SYNOPSIS
#include <limits.h>

DESCRIPTION
The <limits.h> header defines various symbolic names. Different categories of names are described in the tables below.

The names represent various limits on resources that the system imposes on applications.

Implementations may choose any appropriate value for each limit, provided it is not more restrictive than the values listed in the column headed Minimum Acceptable Value in the table below. Symbolic constant names beginning with _POSIX may be found in <unistd.h>.

Applications should not assume any particular value for a limit. To achieve maximum portability, an application should not require more resource than the quantity listed in the Minimum Acceptable Value column. However, an application wishing to avail itself of the full amount of a resource available on an implementation may make use of the value given in <limits.h> on that particular system, by using the symbolic names listed in the first column of the table. It should be noted, however, that many of the listed limits are not invariant, and at run time, the value of the limit may differ from those given in this header, for the following reasons:

- The limit is pathname-dependent.
- The limit differs between the compile and run-time machines.

For these reasons, an application may use the fpathconf(), pathconf() and sysconf() functions to determine the actual value of a limit at run time.

The items in the list ending in _MIN give the most negative values that the mathematical types are guaranteed to be capable of representing. Numbers of a more negative value may be supported on some systems, as indicated by the <limits.h> header on the system, but applications requiring such numbers are not guaranteed to be portable to all systems.

The symbol * in the Minimum Acceptable Value column indicates that there is no guaranteed value across all XSI-conformant systems.
Run-time Invariant Values (Possibly Indeterminate)

A definition of one of the symbolic names in the following table will be omitted from `<limits.h>` on specific implementations where the corresponding value is equal to or greater than the stated minimum, but is indeterminate.

This might depend on the amount of available memory space on a specific instance of a specific implementation. The actual value supported by a specific instance will be provided by the `sysconf()` function.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Minimum Acceptable Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARG_MAX</td>
<td>Maximum length of argument to the <code>exec</code> functions including environment data.</td>
<td></td>
</tr>
<tr>
<td>AEXIT_MAX</td>
<td>Maximum number of functions that may be registered with <code>atexit()</code></td>
<td>32</td>
</tr>
<tr>
<td>CHILD_MAX</td>
<td>Maximum number of simultaneous processes per real user ID.</td>
<td>25</td>
</tr>
<tr>
<td>IOV_MAX</td>
<td>Maximum number of <code>iovec</code> structures that one process has available for use with <code>readv()</code> or <code>writev()</code></td>
<td>_XOPENIOV_MAX</td>
</tr>
<tr>
<td>OPEN_MAX</td>
<td>Maximum number of files that one process can have open at any one time.</td>
<td>20</td>
</tr>
<tr>
<td>PAGESIZE</td>
<td>Size in bytes of a page.</td>
<td>1</td>
</tr>
<tr>
<td>PAGE_SIZE</td>
<td>Same as PAGESIZE. If either PAGESIZE or PAGE_SIZE is defined, the other will be defined with the same value.</td>
<td></td>
</tr>
<tr>
<td>PASS_MAX</td>
<td>Maximum number of significant bytes in a password (not including terminating null). (TO BE WITHDRAWN)</td>
<td>8</td>
</tr>
<tr>
<td>STREAM_MAX</td>
<td>The number of streams that one process can have open at one time. If defined, it has the same value as <code>[FOPEN_MAX]</code> (see <code>&lt;stdio.h&gt;</code>).</td>
<td>_POSIX_STREAM_MAX</td>
</tr>
<tr>
<td>TZNAME_MAX</td>
<td>Maximum number of bytes supported for the name of a time zone (not of the TZ variable).</td>
<td>_POSIX_TZNAME_MAX</td>
</tr>
</tbody>
</table>
Pathname Variable Values

The values in the following table may be constants within an implementation or may vary from one pathname to another. For example, file systems or directories may have different characteristics.

A definition of one of the values will be omitted from the `<limits.h>` on specific implementations where the corresponding value is equal to or greater than the stated minimum, but where the value can vary depending on the file to which it is applied. The actual value supported for a specific pathname will be provided by the `pathconf()` function.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Minimum Acceptable Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK_MAX</td>
<td>Maximum number of links to a single file.</td>
<td>_POSIX_LINK_MAX</td>
</tr>
<tr>
<td>MAX_CANON</td>
<td>Maximum number of bytes in a terminal canonical input line.</td>
<td>_POSIX_MAX_CANON</td>
</tr>
<tr>
<td>MAX_INPUT</td>
<td>Minimum number of bytes for which space will be available in a terminal input queue; therefore, the maximum number of bytes a portable application may require to be typed as input before reading them.</td>
<td>_POSIX_MAX_INPUT</td>
</tr>
<tr>
<td>NAME_MAX</td>
<td>Maximum number of bytes in a filename (not including terminating null).</td>
<td>_POSIX_NAME_MAX</td>
</tr>
<tr>
<td>PATH_MAX</td>
<td>Maximum number of bytes in a pathname, including the terminating null character.</td>
<td>_POSIX_PATH_MAX</td>
</tr>
<tr>
<td>PIPE_BUF</td>
<td>Maximum number bytes that is guaranteed to be atomic when writing to a pipe.</td>
<td>_POSIX_PIPE_BUF</td>
</tr>
</tbody>
</table>
Run-time Increasable Values

The magnitude limitations in the following table will be fixed by specific implementations. An application should assume that the value supplied by `<limits.h>` in a specific implementation is the minimum that pertains whenever the application is run under that implementation. A specific instance of a specific implementation may increase the value relative to that supplied by `<limits.h>` for that implementation. The actual value supported by a specific instance will be provided by the `sysconf()` function.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC_BASE_MAX</td>
<td>Maximum obase values allowed by the <code>bc</code> utility.</td>
</tr>
<tr>
<td>BC_DIM_MAX</td>
<td>Maximum number of elements permitted in an array by the <code>bc</code> utility.</td>
</tr>
<tr>
<td>BC_SCALE_MAX</td>
<td>Maximum scale value allowed by the <code>bc</code> utility.</td>
</tr>
<tr>
<td>BC_STRING_MAX</td>
<td>Maximum length of a string constant accepted by the <code>bc</code> utility.</td>
</tr>
<tr>
<td>COLL_WEIGHTS_MAX</td>
<td>Maximum number of weights that can be assigned to an entry of the <code>LC_COLLATE</code> order keyword in the locale definition file; see the XBD specification, Chapter 5, Locale.</td>
</tr>
<tr>
<td>EXPR_NEST_MAX</td>
<td>Maximum number of expressions that can be nested within parentheses by the <code>expr</code> utility.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LINE_MAX</td>
<td>Unless otherwise noted, the _POSIX2_LINE_MAX maximum length, in bytes, of a utility’s input line (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing newline.</td>
</tr>
<tr>
<td>NGROUPS_MAX</td>
<td>Maximum number of simultaneous supplementary group IDs per process.</td>
</tr>
<tr>
<td>RE_DUP_MAX</td>
<td>Maximum number of repeated _POSIX2_RE_DUP_MAX occurrences of a regular expression permitted when using the interval notation (m, n); see the XBD specification, Chapter 7, Regular Expressions.</td>
</tr>
</tbody>
</table>

### Minimum Values

The symbolic constants in the following table are defined in `<limits.h>` with the values shown. These are symbolic names for the most restrictive value for certain features on a system conforming to this document. Related symbolic constants are defined elsewhere in this document which reflect the actual implementation and which need not be as restrictive. A conforming implementation will provide values at least this large. A portable application must not require a larger value for correct operation.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_POSIX_ARG_MAX</td>
<td>Maximum length of argument to the <code>exec</code> functions including environment data.</td>
<td>4096</td>
</tr>
<tr>
<td>_POSIX_CHILD_MAX</td>
<td>Maximum number of simultaneous processes per real user ID.</td>
<td>6</td>
</tr>
<tr>
<td>_POSIX_LINK_MAX</td>
<td>Maximum number of links to a single file.</td>
<td>8</td>
</tr>
<tr>
<td>_POSIX_MAX_CANON</td>
<td>Maximum number of bytes in a terminal canonical input queue.</td>
<td>255</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>_POSIX_MAX_INPUT</td>
<td>Maximum number of bytes allowed in a terminal input queue.</td>
<td>255</td>
</tr>
<tr>
<td>_POSIX_NAME_MAX</td>
<td>Maximum number of bytes in a filename (not including terminating null).</td>
<td>14</td>
</tr>
<tr>
<td>_POSIX_NGROUPS_MAX</td>
<td>Maximum number of simultaneous supplementary group IDs per process.</td>
<td>0</td>
</tr>
<tr>
<td>_POSIX_OPEN_MAX</td>
<td>Maximum number of files that one process can have open at any one time.</td>
<td>16</td>
</tr>
<tr>
<td>_POSIX_PATH_MAX</td>
<td>Maximum number of bytes in a pathname.</td>
<td>255</td>
</tr>
<tr>
<td>_POSIX_PIPE_BUF</td>
<td>Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.</td>
<td>512</td>
</tr>
<tr>
<td>_POSIX_SSIZE_MAX</td>
<td>The value that can be stored in an object of type ssize_t.</td>
<td>32767</td>
</tr>
<tr>
<td>_POSIX_STREAM_MAX</td>
<td>The number of streams that one process can have open at one time.</td>
<td>8</td>
</tr>
<tr>
<td>_POSIX_TZNAME_MAX</td>
<td>Maximum number of bytes supported for the name of a time zone (not of TZ variable).</td>
<td>3</td>
</tr>
<tr>
<td>_POSIX2_BC_BASE_MAX</td>
<td>Maximum obase values allowed by the bc utility.</td>
<td>99</td>
</tr>
<tr>
<td>_POSIX2_BC_DIM_MAX</td>
<td>Maximum number of elements permitted in an array by the bc utility.</td>
<td>2048</td>
</tr>
<tr>
<td>_POSIX2_BC_SCALE_MAX</td>
<td>Maximum scale value allowed by the bc utility.</td>
<td>99</td>
</tr>
<tr>
<td>_POSIX2_BC_STRING_MAX</td>
<td>Maximum length of a string constant accepted by the bc utility.</td>
<td>1000</td>
</tr>
<tr>
<td>_POSIX2_COLL_WEIGHTS_MAX</td>
<td>Maximum number of weights that can be assigned to an entry of the LC_COLLATE order keyword in the locale definition file; see the XBD specification, Chapter 5, Locale.</td>
<td>2</td>
</tr>
<tr>
<td>_POSIX2_EXPR_NEST_MAX</td>
<td>Maximum number of expressions that can be nested within parentheses by the expr utility.</td>
<td>32</td>
</tr>
<tr>
<td>_POSIX2_LINE_MAX</td>
<td>Unless otherwise noted, the maximum length, in bytes, of a utility’s input line (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing newline.</td>
<td>2048</td>
</tr>
<tr>
<td>_POSIX2_RE_DUP_MAX</td>
<td>Maximum number of repeated occurrences of a regular expression permitted when using the interval notation ( {m, n} ); see the XBD specification, Chapter 7, Regular Expressions.</td>
<td>255</td>
</tr>
<tr>
<td>_XOPEN_IOV_MAX</td>
<td>Maximum number of iovec structures that one process has available for use with readv() or writev().</td>
<td>16</td>
</tr>
</tbody>
</table>
Numerical Limits

The values in the following tables are defined in `<limits.h>` and will be constant expressions suitable for use in `#if` preprocessing directives. Moreover, except for CHAR_BIT, DBL_DIG, DBL_MAX, FLT_DIG, FLT_MAX, LONG_BIT, WORD_BIT and MB_LEN_MAX, the symbolic names will be defined as expressions of the correct type.

If the value of an object of type `char` is treated as a signed integer when used in an expression, the value of CHAR_MIN is the same as that of SCHAR_MIN and the value of CHAR_MAX is the same as that of UCHAR_MAX. Otherwise, the value of CHAR_MIN is 0 and the value of CHAR_MAX is the same as that of UCHAR_MAX.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Minimum Acceptable Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR_BIT</td>
<td>Number of bits in a type <code>char</code>.</td>
<td>8</td>
</tr>
<tr>
<td>CHAR_MAX</td>
<td>Maximum value of a type <code>char</code>.</td>
<td>UCHAR_MAX or SCHAR_MAX</td>
</tr>
<tr>
<td>DBL_DIG</td>
<td>Digits of precision of a type <code>double</code>.</td>
<td>10</td>
</tr>
<tr>
<td>DBL_MAX</td>
<td>Maximum value of a type <code>double</code>.</td>
<td>1E+37</td>
</tr>
<tr>
<td>FLT_DIG</td>
<td>Digits of precision of a type <code>float</code>.</td>
<td>6</td>
</tr>
<tr>
<td>FLT_MAX</td>
<td>Maximum value of a type <code>float</code>.</td>
<td>1E+37</td>
</tr>
<tr>
<td>INT_MAX</td>
<td>Maximum value of an <code>int</code>.</td>
<td>32767</td>
</tr>
<tr>
<td>LONG_BIT</td>
<td>Number of bits in a <code>long int</code>.</td>
<td>32</td>
</tr>
<tr>
<td>LONG_MAX</td>
<td>Maximum value of a <code>long int</code>.</td>
<td>+2147483647</td>
</tr>
<tr>
<td>MB_LEN_MAX</td>
<td>Maximum number of bytes in a character, for any supported locale.</td>
<td>1</td>
</tr>
<tr>
<td>SCHAR_MAX</td>
<td>Maximum value of a type signed <code>char</code>.</td>
<td>+127</td>
</tr>
<tr>
<td>SHRT_MAX</td>
<td>Maximum value of a type <code>short</code>.</td>
<td>+32767</td>
</tr>
<tr>
<td>SSIZE_MAX</td>
<td>Maximum value of an object of type <code>size_t</code></td>
<td>_POSIX_SSIZE_MAX</td>
</tr>
<tr>
<td>UCHAR_MAX</td>
<td>Maximum value of a type <code>unsigned char</code>.</td>
<td>255</td>
</tr>
<tr>
<td>UINT_MAX</td>
<td>Maximum value of a type <code>unsigned int</code>.</td>
<td>65535</td>
</tr>
<tr>
<td>ULONG_MAX</td>
<td>Maximum value of a type <code>unsigned long int</code>.</td>
<td>4294967295</td>
</tr>
<tr>
<td>USHRT_MAX</td>
<td>Maximum value for a type <code>unsigned short int</code>.</td>
<td>65535</td>
</tr>
<tr>
<td>WORD_BIT</td>
<td>Number of bits in a word or type <code>int</code>.</td>
<td>16</td>
</tr>
</tbody>
</table>
**<limits.h>**

### Name Description Maximum Acceptable Value

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Maximum Acceptable Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR_MIN</td>
<td>Minimum value of a type char.</td>
<td>SCHAR_MIN or 0</td>
</tr>
<tr>
<td>INT_MIN</td>
<td>Minimum value of a type int.</td>
<td>−32 767</td>
</tr>
<tr>
<td>LONG_MIN</td>
<td>Minimum value of a type long int.</td>
<td>−2 147 483 647</td>
</tr>
<tr>
<td>SCHAR_MIN</td>
<td>Minimum value of a type signed char.</td>
<td>−127</td>
</tr>
<tr>
<td>SHRT_MIN</td>
<td>Minimum value of a type short.</td>
<td>−32 767</td>
</tr>
</tbody>
</table>

### Other Invariant Values

The following constants are defined on all systems in `<limits.h>`.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Minimum Acceptable Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARCLASS_NAME_MAX</td>
<td>Maximum number of bytes in a character class name.</td>
<td>14</td>
</tr>
<tr>
<td>NL_ARGMAX</td>
<td>Maximum value of digit in calls to the <code>printf()</code> and <code>scanf()</code> functions.</td>
<td>9</td>
</tr>
<tr>
<td>NL_LANGMAX</td>
<td>Maximum number of bytes in a LANG name.</td>
<td>14</td>
</tr>
<tr>
<td>NL_MSGMAX</td>
<td>Maximum message number.</td>
<td>32 767</td>
</tr>
<tr>
<td>NL_NMAX</td>
<td>Maximum number of bytes in an N-to-1 collation mapping.</td>
<td>*</td>
</tr>
<tr>
<td>NL_SETMAX</td>
<td>Maximum set number.</td>
<td>255</td>
</tr>
<tr>
<td>NL_TEXTMAX</td>
<td>Maximum number of bytes in a message string.</td>
<td>_POSIX2_LINE_MAX</td>
</tr>
<tr>
<td>NZERO</td>
<td>Default process priority.</td>
<td>20</td>
</tr>
<tr>
<td>TMP_MAX</td>
<td>Minimum number of unique pathnames generated by <code>tmpnam()</code>. Maximum number of times an application can call <code>tmpnam()</code> reliably. (TO BE WITHDRAWN)</td>
<td>10 000</td>
</tr>
</tbody>
</table>

### Withdrawn Constants

The following constants are withdrawn and may no longer be supported.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBL_MIN</td>
<td>WITHDRAWN</td>
</tr>
<tr>
<td>FLT_MIN</td>
<td>WITHDRAWN</td>
</tr>
</tbody>
</table>

### APPLICATION USAGE

TMP_MAX is moved to `<stdio.h>` to align with the ISO C standard.

DBL_DIG, DBL_MAX, DBL_MIN, FLT_DIG, FLT_MAX, and FLT_MIN are moved to `<float.h>`.

DBL_MIN and FLT_MIN are withdrawn since the values previously defined here were in direct conflict with the values in `<float.h>` as required by the ISO C standard.

### SEE ALSO

`fpathconf()`, `pathconf()`, `sysconf()`.
CHANGE HISTORY

First released in Issue 1.

Issue 4

This entry is largely restructured to improve symbol grouping. A great many symbols, too numerous to mention, have also been added for alignment with the ISO POSIX-2 standard.

The following changes are incorporated for alignment with the ISO C standard:

- The constants INT_MIN, LONG_MIN and SHRT_MIN are changed from values ending in 8 to ones ending in 7.
- The DBL_DIG, DBL_MAX, FLT_DIG and FLT_MAX symbols are marked both as extensions and TO BE WITHDRAWN.
- The LONG_BIT and WORD_BIT symbols are marked as extensions.
- The DBL_MIN and FLT_MIN symbols are withdrawn.
- Text introducing numerical limits now indicates that they will be constant expressions suitable for use in #if preprocessing directives.

The following change is incorporated for alignment with the FIPS requirements:

- The minimum acceptable value for NGROUPS_MAX is changed from _POSIX_NGROUPS_MAX to 8. This is marked as an extension.

Other changes are incorporated as follows:

- A sentence is added to the DESCRIPTION section indicating that names beginning with _POSIX can be found in <unistd.h>.
- The PASS_MAX and TMP_MAX symbols are marked TO BE WITHDRAWN.
- Use of the terms “bytes” and “characters” is rationalised to make it clear when the description is referring to either single-byte values or possibly multi-byte characters.
- CHARCLASS_NAME_MAX is added to the table of Other Invariant Values and marked as an extension.

Issue 4, Version 2

The DESCRIPTION is revised for X/OPEN UNIX conformance as follows:

- Under Run-time Invariant Values, ATEXIT_MAX, IOV_MAX, PAGESIZE and PAGE_SIZE are added.
- Under Minimum Values, _XOPEN_IOV_MAX is added.
NAME
locale.h — category macros

SYNOPSIS
#include <locale.h>

DESCRIPTION
The `<locale.h>` header provides a definition for structure `lconv`, which includes at least the following members. (See the definitions of LC_MONETARY in the XBD specification, Section 5.3.3, LC_MONETARY, and the XBD specification, Section 5.3.4, LC_NUMERIC.)

```c
char *currency_symbol
char *decimal_point
char frac_digits
char *grouping
char *int_curr_symbol
char int_frac_digits
char *mon_decimal_point
char *mon_grouping
char *mon_thousands_sep
char *negative_sign
char n_cs_precedes
char n_sep_by_space
char n_sign_posn
char *positive_sign
char p_cs_precedes
char p_sep_by_space
char p_sign_posn
char *thousands_sep
```

The `<locale.h>` header defines NULL (as defined in `<stddef.h>`) and at least the following as macros:

```
LC_ALL
LC_COLLATE
LC_CTYPE
LC_MESSAGES
LC_MONETARY
LC_NUMERIC
LC_TIME
```

which expand to distinct integral-constant expressions, for use as the first argument to the `setlocale()` function.

Additional macro definitions, beginning with the characters LC_ and an upper-case letter, may also be given here.

The following are declared as functions and may also be defined as macros:

```
struct lconv *localeconv (void);
char setlocale(int category, const char *locale);
```

SEE ALSO
`localeconv()`, `setlocale()`, the XBD specification, Chapter 6, Environment Variables.
CHANGE HISTORY

First released in Issue 3.
Entry included for alignment with the ISO C standard.

Issue 4
The following changes are incorporated for alignment with the ISO C standard:

- The function declarations in this header are expanded to full ISO C prototypes.
- The definition of \texttt{struct lconv} is added.
- A reference to \texttt{<stddef.h>} is added for the definition of NULL.
<math.h>

NAME
math.h – mathematical declarations

SYNOPSIS
#include <math.h>

DESCRIPTION
The <math.h> header provides for the following constants. The values are of type double and are accurate within the precision of the double type.

<table>
<thead>
<tr>
<th>EX</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_E</td>
<td>Value of e</td>
</tr>
<tr>
<td>M_LOG2E</td>
<td>Value of log₂e</td>
</tr>
<tr>
<td>M_LOG10E</td>
<td>Value of log₁₀e</td>
</tr>
<tr>
<td>M_LN2</td>
<td>Value of log₂</td>
</tr>
<tr>
<td>M_LN10</td>
<td>Value of log₁₀</td>
</tr>
<tr>
<td>M_PI</td>
<td>Value of π</td>
</tr>
<tr>
<td>M_PI_2</td>
<td>Value of π/2</td>
</tr>
<tr>
<td>M_PI_4</td>
<td>Value of π/4</td>
</tr>
<tr>
<td>M_1_PI</td>
<td>Value of 1/π</td>
</tr>
<tr>
<td>M_2_PI</td>
<td>Value of 2/π</td>
</tr>
<tr>
<td>M_2_SQRTPI</td>
<td>Value of 2/√π</td>
</tr>
<tr>
<td>M_SQRT2</td>
<td>Value of √2</td>
</tr>
<tr>
<td>M_SQRT1_2</td>
<td>Value of 1/√2</td>
</tr>
</tbody>
</table>

The header defines the following symbolic constants:

<table>
<thead>
<tr>
<th>EX</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXFLOAT</td>
<td>Value of maximum non-infinite single-precision floating point number.</td>
</tr>
<tr>
<td>HUGE_VAL</td>
<td>A positive double expression, not necessarily representable as a float. Used as an error value returned by the mathematics library. HUGE_VAL evaluates to +∞ on systems supporting the ANSI/IEEE Std 754:1985 standard.</td>
</tr>
</tbody>
</table>

The following are declared as functions and may also be defined as macros:

double acos(double x);
double asin(double x);
double atan(double x);
double atan2(double y, double x);
double ceil(double x);
double cos(double x);
double cosh(double x);
double exp(double x);
double fabs(double x);
double floor(double x);
double fmod(double x, double y);
double frexp(double value, int *exp);
double ldexp(double value, int exp);
double log(double x);
double log10(double x);
double modf(double value, double *iptr);
double pow(double x, double y);
double sin(double x);
double sinh(double x);
double sqrt(double x);
double tan(double x);
double tanh(double x);

EX
double erf(double x);
double erfc(double x);
double gamma(double x);
double hypot(double x, double y);
double j0(double x);
double j1(double x);
double jn(int n, double x);
double lgamma(double x);
double y0(double x);
double y1(double x);
double yn(int n, double x);
int isnan(double x);

UX
double acosh(double x);
double asinh(double x);
double atanh(double x);
double cbrt(double x);
double expm1(double x);
int ilogb(double x);
double log1p(double x);
double logb(double x);
double nextafter(double x, double y);
double remainder(double x, double y);
double rint(double x);
double scalb(double x, double n);

The following external variable is defined:

EX
extern int signgam;

SEE ALSO
acos(), acosh(), asin(), atan(), atan2(), cbrt(), ceil(), cos(), cosh(), erf(), exp(), expm1(), fabs(), floor(), fmod(), frexp(), gamma(), hypot(), ilogb(), isnan(), j0(), ldexp(), lgamma(), log(), log10(), log1p(), logb(), modf(), nextafter(), pow(), remainder(), rint(), scalb(), sin(), sinh(), sqrt(), tan(),
tanh().

CHANGE HISTORY
First released in Issue 1.

Issue 4
The following changes are incorporated for alignment with the ISO C standard:

- The description of HUGE_VAL is changed to indicate that this value is not necessarily
  representable as a float.
- The function declarations in this header are expanded to full ISO C prototypes.

Other changes are incorporated as follows:

- The constants M_E and MAXFLOAT are marked as extensions.
- The functions declared in this header are subdivided into those defined in the ISO C
  standard, and those defined only by X/Open. Functions in the latter group are marked as
  extensions, as is the external variable signgam.
Issue 4, Version 2

The following change is incorporated for X/OPEN UNIX conformance:

- The `acosh()`, `asinh()`, `atanh()`, `cbrt()`, `expm1()`, `ilogb()`, `log1p()`, `logb()`, `nextafter()`, `remainder()`, `rint()` and `scalb()` functions are added to the list of functions declared in this header.
NAME
monetary.h — monetary types

SYNOPSIS

```c
#include <monetary.h>
```

DESCRIPTION
The `<monetary.h>` header defines the following data types through typedef:

- `size_t` As described in `<stddef.h>`.
- `ssize_t` As described in `<sys/types.h>`.

The following is declared as a function and may also be defined as a macro:

```c
ssize_t strfmon(char *s, size_t maxsize, const char *format, ...);
```

SEE ALSO
strfmon().

CHANGE HISTORY
First released in Issue 4.
NAME

ndbm.h — definitions for ndbm database operations

SYNOPSIS

UX

#include <ndbm.h>

DESCRIPTION

The <ndbm.h> header defines the datum type as a structure that includes at least the following members:

- **void *dptr** — A pointer to the application’s data
- **size_t dsize** — The size of the object pointed to by dptr

The <ndbm.h> header defines the DBM type through typedef.

The following constants are defined as possible values for the store_mode argument to dbm_store():

- **DBM_INSERT** — Insertion of new entries only
- **DBM_REPLACE** — Allow replacing existing entries

The following are declared as functions and may also be defined as macros:

```c
int dbm_clearerr(DBM *db);
void dbm_close(DBM *db);
int dbm_delete(DBM *db, datum key);
int dbm_error(DBM *db);
datum dbm_fetch(DBM *db, datum key);
datum dbm_firstkey(DBM *db);
datum dbm_nextkey(DBM *db);
DBM *dbm_open(const char *file, int open_flags, mode_t file_mode);
int dbm_store(DBM *db, datum key, datum content, int store_mode);
```

SEE ALSO

dbm_clearerr().

CHANGE HISTORY

First released in Issue 4, Version 2.
**NAME**

nl_types.h — data types

**SYNOPSIS**

```c
#include <nl_types.h>
```

**DESCRIPTION**

The `<nl_types.h>` header contains definitions of at least the following types:

- **nl_catd**
  Used by the message catalogue functions `catopen()`, `catgets()` and `catclose()` to identify a catalogue descriptor.

- **nl_item**
  Used by `nl_langinfo()` to identify items of `langinfo` data. Values of objects of type `nl_item` are defined in `<langinfo.h>`.

The `<nl_types.h>` header contains definitions of at least the following constants:

- **NL_SETD**
  Used by `gencat` when no `$set` directive is specified in a message text source file, see the *Internationalisation Guide, Chapter 3, The Message System*. This constant can be passed as the value of `set_id` on subsequent calls to `catgets()` (that is, to retrieve messages from the default message set). The value of NL_SETD is implementation-dependent.

- **NL_CAT_LOCALE**
  Value that must be passed as the `oflag` argument to `catopen()` to ensure that message catalogue selection depends on the LC_MESSAGES locale category, rather than directly on the LANG environment variable.

The following are declared as functions and may also be defined as macros:

```c
int catclose(nl_catd catd);
char *catgets(nl_catd catd, int set_id, int msg_id, const char * s);
nl_catd catopen(const char * name, int oflag);
```

**SEE ALSO**

`catclose()`, `catgets()`, `catopen()`, `nl_langinfo()`, `<langinfo.h>`, the XCU specification, `gencat`.

**CHANGE HISTORY**

First released in Issue 2.

**Issue 4**

The following change is incorporated for alignment with the ISO C standard:

- The function declarations in this header are expanded to full ISO C prototypes.
NAME
poll.h — definitions for the poll() function

SYNOPSIS

```
#include <poll.h>
```

DESCRIPTION

The `<poll.h>` header defines the `pollfd` structure that includes at least the following member:

```
int fd the following descriptor being polled
short int events the input event flags (see below)
short int revents the output event flags (see below)
```

The `<poll.h>` header defines the following type through typedef:

```
nfds_t An unsigned integral type used for the number of file descriptors.
```

The following symbolic constants are defined, zero or more of which may be OR-ed together to form the `events` or `revents` members in the `pollfd` structure:

- `POLLIN` Same value as `POLLRDNORM | POLLRDBAND`.
- `POLLRDNORM` Data on priority band 0 may be read.
- `POLLRDBAND` Data on priority bands greater than 0 may be read.
- `POLLPRI` High priority data may be read.
- `POLLOUT` Same value as `POLLWRNORM`.
- `POLLWRNORM` Data on priority band 0 may be written.
- `POLLERR` An error has occurred (`revents` only).
- `POLLHUP` Device has been disconnected (`revents` only).
- `POLLNVAL` Invalid `fd` member (`revents` only).

The `<poll.h>` header declares the following function which may also be defined as a macro:

```
int poll(struct pollfd fds[], nfds_t nfds, int timeout);
```

SEE ALSO

poll().

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
pwd.h — password structure

SYNOPSIS
#include <pwd.h>

DESCRIPTION
The <pwd.h> header provides a definition for struct passwd, which includes the following members:

```c
char *pw_name  user's login name
uid_t pw_uid   numerical user ID
gid_t pw_gid   numerical group ID
char *pw_dir   initial working directory
char *pw_shell program to use as shell
```

EX
The gid_t and uid_t types are defined as described in `<sys/types.h>`.

The following are declared as functions and may also be defined as macros:

```c
struct passwd *getpwnam(const char *name);
struct passwd *getpwuid(uid_t uid);
```

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

APPLICATION USAGE
The passwd structure may contain additional members.

SEE ALSO
depwent(), getpwnam(), getpwuid(), `<sys/types.h>`.

CHANGE HISTORY
First released in Issue 1.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:
- The function declarations in this header are expanded to full ISO C prototypes.

Another change is incorporated as follows:
- Reference to the header `<sys/types.h>` is added for the definitions of gid_t and uid_t. This is marked as an extension.

Issue 4, Version 2
For X/OPEN UNIX conformance, the getpwent(), endpwent() and setpwent() functions are added to the list of functions declared in this header.
NAME
regex.h — regular-expression-matching types

SYNOPSIS
#include <regex.h>

DESCRIPTION
The <regex.h> header defines the structures and symbolic constants used by the regcomp(),
regexec(), regerror() and regfree() functions.

The structure type regex_t contains at least the following member:

size_t re_nsub number of parenthesised subexpressions

The type regoff_t is defined as a signed arithmetic type that can hold the largest value that can
be stored in either a type off_t or type ssize_t. The structure type regmatch_t contains at least
the following members:

regoff_t rm_so byte offset from start of string to start of substring
regoff_t rm_eo byte offset from start of string of the first character after the end of
substring

Values for the cflags parameter to the regcomp() function:
REG_EXTENDED Use Extended Regular Expressions.
REG_ICASE Ignore case in match.
REG_NOSUB Report only success or fail in regexec().
REG_NEWLINE Change the handling of newline.

Values for the eflags parameter to the regexec() function:
REG_NOTBOL The circumflex character (^), when taken as a special character, will not
match the beginning of string.
REG_NOTEOL The dollar sign ($), when taken as a special character, will not match the
end of string.

The following constants are defined as error return values:
REG_NOMATCH regexec() failed to match.
REG_BADPAT Invalid regular expression.
REG_ECOLLATE Invalid collating element referenced.
REG_ECTYPE Invalid character class type referenced.
REG_EESCAPE Trailing \ in pattern.
REG_ESUBREG Number in \digit invalid or in error.
REG_EBRACK [ ] imbalance.
REG_EPAREN \( \) or ( ) imbalance.
REG_EBRACE \{ \} imbalance.
REG_BADBR Content of \{ \} invalid: not a number, number too large, more than two
numbers, first larger than second.
REG_ERANGE Invalid endpoint in range expression.
REG_ESPACE Out of memory.
REG_BADRPT ?, * or + not preceded by valid regular expression.
REG_ENOSYS The implementation does not support the function.
The following are declared as functions and may also be declared as macros:

```c
int regcomp(regex_t *preg, const char *pattern, int cflags);
int regexec(const regex_t *preg, const char *string,
            size_t nmatch, regmatch_t pmatch[], int eflags);
size_t regerror(int errcode, const regex_t *preg,
                char *errbuf, size_t errbuf_size);
void regfree(regex_t *preg);
```

The implementation may define additional macros or constants using names beginning with REG_.

SEE ALSO
regcomp(), regex(), the XCU specification.

CHANGE HISTORY
First released in Issue 4.
Originally derived from the ISO POSIX-2 standard.
NAME
re_comp.h — regular-expression-matching functions for re_comp() (TO BE WITHDRAWN)

SYNOPSIS
```c
#include <re_comp.h>
```

DESCRIPTION
The following are declared as functions and may also be declared as macros:

```c
char  *re_comp(const char *string);
int    re_exec(const char *string);
```

SEE ALSO
re_comp().

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME

regexp.h — regular-expression declarations (TO BE WITHDRAWN)

SYNOPSIS

```c
#include <regexp.h>
```

DESCRIPTION

In the `<regexp.h>` header, each of the following is declared as a function, or defined as a macro, or both:

```c
int advance(const char *string, const char *expbuf);
char *compile(char *instring, char *expbuf, const char *endbuf,
    int eof);
int step(const char *string, const char *expbuf);
```

and the following are declared as external variables:

```c
extern char *loc1;
extern char *loc2;
extern char *locs;
```

SEE ALSO

`regexp()`.

CHANGE HISTORY

First released in Issue 3.

Entry derived from System V Release 2.0.

Issue 4

The following changes are incorporated in this issue:

- The function declarations in this header are expanded to full ISO C prototypes.
- The interface is marked TO BE WITHDRAWN.
NAME

search.h — search tables

SYNOPSIS

```c
#include <search.h>
```

DESCRIPTION

The `<search.h>` header provides a type definition, `ENTRY`, for structure `entry` which includes the following members:

```c
char *key
void *data
```

and defines `ACTION` and `VISIT` as enumeration data types through type definitions as follows:

```c
enum { FIND, ENTER } ACTION;
enum { preorder, postorder, endorder, leaf } VISIT;
```

The `size_t` type is defined as described in `<sys/types.h>`.

Each of the following is declared as a function, or defined as a macro, or both:

- `int hcreate(size_t nel);`
- `void hdestroy(void);`
- `ENTRY *hsearch(ENTRY item, ACTION action);`
- `void insque(void *element, void *pred);`
- `void *lfind(const void *key, const void *base, size_t *nelp, size_t width, int (*compar)(const void *, const void *));`
- `void *lsearch(const void *key, void *base, size_t *nelp, size_t width, int (*compar)(const void *, const void *));`
- `void remque(void *element);`
- `void *tdelete(const void *key, void **rootp, int (*compar)(const void *, const void *));`
- `void *tfind(const void *key, void *const *rootp, int (*compar)(const void *, const void *));`
- `void *tsearch(const void *key, void **rootp, int (*compar)(const void *, const void *));`
- `void twalk(const void *root, void (*action)(const void *, VISIT, int ));`

SEE ALSO

`hsearch()`, `insque()`, `lsearch()`, `remque()`, `tsearch()`, `<sys/types.h>`.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated in this issue:

- The function declarations in this header are expanded to full ISO C prototypes.
- Reference to the header `<sys/types.h>` is added for the definition of `size_t`.

Issue 4, Version 2

For X/Open UNIX conformance, the `insque()` and `remque()` functions are added to the list of functions declared in this header.
NAME

setjmp.h — stack environment declarations

SYNOPSIS

#include <setjmp.h>

DESCRIPTION

The <setjmp.h> header contains the type definitions for array types jmp_buf and sigjmp_buf.

The following are declared as functions and may also be defined as macros:

void longjmp(jmp_buf env, int val);
void siglongjmp(sigjmp_buf env, int val);
UX void _longjmp(jmp_buf env, int val);

Each of the following may be declared as a function, or defined as a macro, or both:

int setjmp(jmp_buf env);
int sigset jmp(sigjmp_buf env, int savemask);
UX int _set jmp(jmp_buf env);

SEE ALSO

longjmp(), _longjmp(), setjmp(), siglongjmp(), sigsetjmp().

CHANGE HISTORY

First released in Issue 1.

Issue 4

The following changes are incorporated for alignment with the ISO C standard:

• The function declarations in this header are expanded to full ISO C prototypes.

• The DESCRIPTION section is changed to indicate that all functions in this header can also be declared as macros.

• The arguments jmp_buf and sigjmp_buf are specified as array types.

Issue 4, Version 2

For X/OPEN UNIX conformance, the _longjmp() and _setjmp() functions are added to the list of functions declared in this header.
NAME

signal.h — signals

SYNOPSIS

#include <signal.h>

DESCRIPTION

The <signal.h> header defines the following symbolic constants, each of which expands to a distinct constant expression of the type:

    void (*)(int)

whose value matches no declarable function.

SIG_DFL Request for default signal handling.
SIG_ERR Return value from signal() in case of error.
SIG_HOLD Request that signal be held.
SIG_IGN Request that signal be ignored.

The following data types are defined through typedef:

    sig_atomic_t        Integral type of an object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts
    sigset_t            Integral or structure type of an object used to represent sets of signals.

    pid_t               As described in <sys/types.h>.

This header also declares the constants that are used to refer to the signals that occur in the system. Signals defined here begin with the letters SIG. Each of the signals have distinct positive integral values. The value 0 is reserved for use as the null signal (see kill()). Additional implementation-dependent signals may occur in the system.
The following signals are supported on all implementations (default actions are explained below the table):

<table>
<thead>
<tr>
<th>Signal</th>
<th>Default Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGABRT</td>
<td>ii</td>
<td>Process abort signal.</td>
</tr>
<tr>
<td>SIGALRM</td>
<td>i</td>
<td>Alarm clock.</td>
</tr>
<tr>
<td>SIGFPE</td>
<td>ii</td>
<td>Erroneous arithmetic operation.</td>
</tr>
<tr>
<td>SIGHUP</td>
<td>i</td>
<td>Hangup.</td>
</tr>
<tr>
<td>SIGILL</td>
<td>ii</td>
<td>Illegal instruction.</td>
</tr>
<tr>
<td>SIGINT</td>
<td>i</td>
<td>Terminal interrupt signal.</td>
</tr>
<tr>
<td>SIGKILL</td>
<td>i</td>
<td>Kill (cannot be caught or ignored).</td>
</tr>
<tr>
<td>SIGPIPE</td>
<td>i</td>
<td>Write on a pipe with no one to read it.</td>
</tr>
<tr>
<td>SIGQUIT</td>
<td>ii</td>
<td>Terminal quit signal.</td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>ii</td>
<td>Invalid memory reference.</td>
</tr>
<tr>
<td>SIGTERM</td>
<td>i</td>
<td>Termination signal.</td>
</tr>
<tr>
<td>SIGUSR1</td>
<td>i</td>
<td>User-defined signal 1.</td>
</tr>
<tr>
<td>SIGUSR2</td>
<td>i</td>
<td>User-defined signal 2.</td>
</tr>
<tr>
<td>SIGCHLD</td>
<td>iii</td>
<td>Child process terminated or stopped.</td>
</tr>
<tr>
<td>SIGCONT</td>
<td>v</td>
<td>Continue executing, if stopped.</td>
</tr>
<tr>
<td>SIGSTOP</td>
<td>iv</td>
<td>Stop executing (cannot be caught or ignored).</td>
</tr>
<tr>
<td>SIGTSTP</td>
<td>iv</td>
<td>Terminal stop signal.</td>
</tr>
<tr>
<td>SIGTIN</td>
<td>iv</td>
<td>Background process attempting read.</td>
</tr>
<tr>
<td>SIGTTOU</td>
<td>iv</td>
<td>Background process attempting write.</td>
</tr>
<tr>
<td>SIGBUS</td>
<td>ii</td>
<td>Bus error.</td>
</tr>
<tr>
<td>SIGPOLL</td>
<td>i</td>
<td>Pollable event.</td>
</tr>
<tr>
<td>SIGPROF</td>
<td>i</td>
<td>Profiling timer expired.</td>
</tr>
<tr>
<td>SIGSYS</td>
<td>ii</td>
<td>Bad system call.</td>
</tr>
<tr>
<td>SIGTRAP</td>
<td>ii</td>
<td>Trace/breakpoint trap.</td>
</tr>
<tr>
<td>SIGURG</td>
<td>i</td>
<td>High bandwidth data is available at a socket.</td>
</tr>
<tr>
<td>SIGTALRM</td>
<td>i</td>
<td>Virtual timer expired.</td>
</tr>
<tr>
<td>SIGVTALRM</td>
<td>i</td>
<td>CPU time limit exceeded.</td>
</tr>
<tr>
<td>SIGXCPU</td>
<td>ii</td>
<td>File size limit exceeded.</td>
</tr>
</tbody>
</table>

The default actions are as follows:

i Abnormal termination of the process. The process is terminated with all the consequences of _exit( ) except that the status is made available to wait( ) and waitpid( ) indicates abnormal termination by the specified signal.

ii Abnormal termination of the process.

EX Additionally, implementation-dependent abnormal termination actions, such as creation of a core file, may occur.

iii Ignore the signal.

iv Stop the process.

v Continue the process, if it is stopped; otherwise ignore the signal.
The header provides a declaration of **struct sigaction**, including at least the following members:

```c
void (*sa_handler)(int) what to do on receipt of signal
sigset_t sa_mask set of signals to be blocked during execution of the
                          signal handling function
int sa_flags special flags
ux void (*)(int, siginfo_t *, void *) sa_sigaction pointer to signal handler function
```

The storage occupied by *sa_handler* and *sa_sigaction* may overlap, and a portable program must not use both simultaneously.

The following are declared as constants:

- **SA_NOCLDSTOP** Do not generate SIGCHLD when children stop.
- **SIG_BLOCK** The resulting set is the union of the current set and the signal set pointed to by the argument *set*.
- **SIG_UNBLOCK** The resulting set is the intersection of the current set and the complement of the signal set pointed to by the argument *set*.
- **SIG_SETMASK** The resulting set is the signal set pointed to by the argument *set*.
- **SA_ONSTACK** Causes signal delivery to occur on an alternate stack.
- **SA_RESETHAND** Causes signal dispositions to be set to SIG_DFL on entry to signal handlers.
- **SA_RESTART** Causes certain functions to become restartable.
- **SA_SIGINFO** Causes extra information to be passed to signal handlers at the time of receipt of a signal.
- **SA_NOCLDWAIT** Causes implementations not to create zombie processes on child death.
- **SA_NODEFER** Causes signal not to be automatically blocked on entry to signal handler.
- **SS_ONSTACK** Process is executing on an alternate signal stack.
- **SS_DISABLE** Alternate signal stack is disabled.
- **MINSIGSTKSZ** Minimum stack size for a signal handler.
- **SIGSTKSZ** Default size in bytes for the alternate signal stack.

The ucontext_t structure is defined through typedef as described in `<ucontext.h>`.

The `<signal.h>` header defines the stack_t type as a structure that includes at least the following members:

```c
void *ss_sp stack base or pointer
size_t ss_size stack size
int ss_flags flags
```

The `<signal.h>` header defines the sigstack structure that includes at least the following members:

```c
int ss_onstack non-zero when signal stack is in use
void *ss_sp signal stack pointer
```
The `<signal.h>` header defines the `siginfo_t` type as a structure that includes at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>si_signo</td>
<td>signal number</td>
</tr>
<tr>
<td>si_errno</td>
<td>if non-zero, an <code>errno</code> value associated with this signal, as defined in <code>&lt;errno.h&gt;</code></td>
</tr>
<tr>
<td>si_code</td>
<td>signal code</td>
</tr>
<tr>
<td>si_pid</td>
<td>sending process ID</td>
</tr>
<tr>
<td>si_uid</td>
<td>real user ID of sending process</td>
</tr>
<tr>
<td>si_addr</td>
<td>address of faulting instruction</td>
</tr>
<tr>
<td>si_status</td>
<td>exit value or signal</td>
</tr>
<tr>
<td>si_band</td>
<td>band event for SIGPOLL</td>
</tr>
</tbody>
</table>
The macros specified in the **Code** column of the following table are defined for use as values of **si_code** that are signal-specific reasons why the signal was generated.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGILL</td>
<td>ILL_ILLOPC</td>
<td>illegal opcode</td>
</tr>
<tr>
<td></td>
<td>ILL_ILLOPN</td>
<td>illegal operand</td>
</tr>
<tr>
<td></td>
<td>ILL_ILLADR</td>
<td>illegal addressing mode</td>
</tr>
<tr>
<td></td>
<td>ILL_ILLTRP</td>
<td>illegal trap</td>
</tr>
<tr>
<td></td>
<td>ILL_PRVOPC</td>
<td>privileged opcode</td>
</tr>
<tr>
<td></td>
<td>ILL_PRVREG</td>
<td>privileged register</td>
</tr>
<tr>
<td></td>
<td>ILL_COPROC</td>
<td>coprocessor error</td>
</tr>
<tr>
<td></td>
<td>ILL_BADSTK</td>
<td>internal stack error</td>
</tr>
<tr>
<td>SIGFPE</td>
<td>FPE_INTDIV</td>
<td>integer divide by zero</td>
</tr>
<tr>
<td></td>
<td>FPE_INTOVF</td>
<td>integer overflow</td>
</tr>
<tr>
<td></td>
<td>FPE_FLTDIV</td>
<td>floating point divide by zero</td>
</tr>
<tr>
<td></td>
<td>FPE_FLTOVF</td>
<td>floating point overflow</td>
</tr>
<tr>
<td></td>
<td>FPE_FLTUND</td>
<td>floating point underflow</td>
</tr>
<tr>
<td></td>
<td>FPE_FLTRES</td>
<td>floating point inexact result</td>
</tr>
<tr>
<td></td>
<td>FPE_FLTINV</td>
<td>invalid floating point operation</td>
</tr>
<tr>
<td></td>
<td>FPE_FLTSUB</td>
<td>subscript out of range</td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>SEGV_MAPERR</td>
<td>address not mapped to object</td>
</tr>
<tr>
<td></td>
<td>SEGV_ACCERR</td>
<td>invalid permissions for mapped object</td>
</tr>
<tr>
<td>SIGBUS</td>
<td>BUS_ADRALN</td>
<td>invalid address alignment</td>
</tr>
<tr>
<td></td>
<td>BUS_ADRERR</td>
<td>non-existent physical address</td>
</tr>
<tr>
<td></td>
<td>BUS_OBJERR</td>
<td>object specific hardware error</td>
</tr>
<tr>
<td>SIGTRAP</td>
<td>TRAP_BRKPT</td>
<td>process breakpoint</td>
</tr>
<tr>
<td></td>
<td>TRAP_TRACE</td>
<td>process trace trap</td>
</tr>
<tr>
<td>SIGCHLD</td>
<td>CLD_EXITED</td>
<td>child has exited</td>
</tr>
<tr>
<td></td>
<td>CLD_KILLED</td>
<td>child has terminated abnormally and did not create a core file</td>
</tr>
<tr>
<td></td>
<td>CLD_DUMPED</td>
<td>child has terminated abnormally and created a core file</td>
</tr>
<tr>
<td></td>
<td>CLD_KILLED</td>
<td>child was killed</td>
</tr>
<tr>
<td></td>
<td>CLD_DUMPED</td>
<td>child has terminated abnormally</td>
</tr>
<tr>
<td></td>
<td>CLD_TRAPPED</td>
<td>traced child has trapped</td>
</tr>
<tr>
<td></td>
<td>CLD_STOPPED</td>
<td>child has stopped</td>
</tr>
<tr>
<td></td>
<td>CLD_CONTINUED</td>
<td>stopped child has continued</td>
</tr>
<tr>
<td>SIGPOLL</td>
<td>POLL_IN</td>
<td>data input available</td>
</tr>
<tr>
<td></td>
<td>POLL_OUT</td>
<td>output buffers available</td>
</tr>
<tr>
<td></td>
<td>POLL_MSG</td>
<td>input message available</td>
</tr>
<tr>
<td></td>
<td>POLL_ERR</td>
<td>I/O error</td>
</tr>
<tr>
<td></td>
<td>POLL_PRI</td>
<td>high priority input available</td>
</tr>
<tr>
<td></td>
<td>POLL_HUP</td>
<td>device disconnected</td>
</tr>
</tbody>
</table>

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 will not generate a different value from the ones described in this list for circumstances described in this list.
In addition, the following signal-specific information will be available:

<table>
<thead>
<tr>
<th>Signal</th>
<th>Member</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGILL</td>
<td>void * si_addr</td>
<td>address of faulting instruction</td>
</tr>
<tr>
<td>SIGFPE</td>
<td>void * si_addr</td>
<td>address of faulting memory reference</td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>void * si_addr</td>
<td>address of faulting memory reference</td>
</tr>
<tr>
<td>SIGBUS</td>
<td>pid_t si_pid</td>
<td>child process ID</td>
</tr>
<tr>
<td></td>
<td>int si_status</td>
<td>exit value or signal</td>
</tr>
<tr>
<td></td>
<td>uid_t si_uid</td>
<td>real user ID of the process that sent the signal</td>
</tr>
<tr>
<td>SIGCHLD</td>
<td>long si_band</td>
<td>band event for POLL_IN, POLL_OUT, or POLL_MSG</td>
</tr>
</tbody>
</table>

For some implementations, the value of si_addr may be inaccurate.

The following are declared as functions and may also be defined as macros:

```c
void (*bsd_signal(int sig, void (*)(int)))(int);
int kill(pid_t pid, int sig);
int killpg(pid_t pgrp, int sig);
int raise(int sig);
int sigaction(int sig, const struct sigaction *act, struct sigaction *oact);
int sigaddset(sigset_t *set, int signo);
int sigdelset(sigset_t *set, int signo);
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sighold(int sig);
int sigignore(int sig);
int siginterrupt(int sig, int flag);
int sigismember(const sigset_t *set, int signo);
int sigmask(int signum);
void (*signal(int sig, void (*)(int)))(int);
int sigpause(int sig);
int sigpending(const sigset_t *set);
int sigprocmask(int how, const sigset_t *set, sigset_t *oset);
int sigrelse(int sig);
void (*sigset(int sig, void (*)(int)))(int);
int sigstack(struct sigstack *ss, struct sigstack *oss);  // (TO BE WITHDRAWN)
int sigsuspend(const sigset_t *sigmask);
```

**SEE ALSO**

alarm(), bsd_signal(), ioctl(), kill(), killpg(), raise(), sigaction(), sigaddset(), sigaltstack(), sigdelset(), sigemptyset(), sigfillset(), siginterrupt(), sigismember(), signal(), sigpending(), sigprocmask(), sigstack(), sigsuspend(), wait(), waitid(), <errno.h>, <streams.h>, <sys/types.h>, <ucontext.h>.

**CHANGE HISTORY**

First released in Issue 1.

**Issue 4**

The following changes are incorporated for alignment with the ISO POSIX-1 standard:
The function declarations in this header are expanded to full ISO C prototypes.

- The DESCRIPTION section is changed:
  - to define the type `sig_atomic_t`
  - to define the syntax of signal names and functions
  - to combine the two tables of constants
  - SIGFPE is no longer limited to floating-point exceptions, but covers all erroneous arithmetic operations.

The following change is incorporated for alignment with the ISO C standard:

- The `raise()` function is added to the list of functions declared in this header.

Other changes are incorporated as follows:

- A reference to `<sys/types.h>` is added for the definition of `pid_t`. This is marked as an extension.

- In the list of signals starting with SIGCHLD, the statement “but a system not supporting the job control option is not obliged to support the functionality of these signals” is removed. This is because job control is defined as mandatory on Issue 4 conforming implementations.

- Reference to implementation-dependent abnormal termination routines, such as creation of a core file, in item ii in the defaults action list is marked as an extension.

### Issue 4, Version 2

The following changes are incorporated for X/OPEN UNIX conformance:

- The SIGTRAP, SIGBUS, SIGSYS, SIGPOLL, SIGPROF, SIGXCPU, SIGXFSZ, SIGURG and SIGVTALRM signals are added to the list of signals that will be supported on all conforming implementations.

- The `sa_sigaction` member is added to the `sigaction` structure, and a note is added that the storage used by `sa_handler` and `sa_sigaction` may overlap.

- The SA_ONSTACK, SA_RESETHAND, SA_RESTART, SA_SIGINFO, SA_NOCLEWDWAIT, SS_ONSTACK, SS_DISABLE, MINSIGSTKSZ and SIGSTKSZ constants are defined. The `stack_t`, `sigstack` and `siginfo` structures are defined.

- Definitions are given for the `ucontext_t`, `stack_t`, `sigstack` and `siginfo_t` types.

- A table is provided listing macros that are defined as signal-specific reasons why a signal was generated. Signal-specific additional information is specified.

- The `bsd_signal()`, `killpg()`, `_longjmp()`, `_setjmp()`, `sigaltstack()`, `sighold()`, `sigignore()`, `siginterrupt()`, `sigpause()`, `sigrese()`, `sigset()` and `sigstack()` functions are added to the list of functions declared in this header.
NAME
stdarg.h — handle variable argument list

SYNOPSIS
#include <stdarg.h>

void va_start(va_list ap , argN);
type va_arg(va_list ap , type);
void va_end(va_list ap);

DESCRIPTION
The <stdarg.h> header contains a set of macros which allows portable functions that accept
variable argument lists to be written. Functions that have variable argument lists (such as
printf()) but do not use these macros are inherently non-portable, as different systems use
different argument-passing conventions.

The type va_list is defined for variables used to traverse the list.

The va_start() macro is invoked to initialise ap to the beginning of the list before any calls to
va_arg().

The object ap may be passed as an argument to another function; if that function invokes the
va_arg() macro with parameter ap, the value of ap in the calling function is indeterminate and
must be passed to the va_end() macro prior to any further reference to ap. The parameter argN is
the identifier of the rightmost parameter in the variable parameter list in the function definition
(the one just before the , . . .). If the parameter argN is declared with the register storage class,
with a function type or array type, or with a type that is not compatible with the type that results
after application of the default argument promotions, the behaviour is undefined.

The va_arg() macro will return the next argument in the list pointed to by ap. Each invocation of
va_arg() modifies ap so that the values of successive arguments are returned in turn. The type
parameter is the type the argument is expected to be. This is the type name specified such that
the type of a pointer to an object that has the specified type can be obtained simply by suffixing a
* to type. Different types can be mixed, but it is up to the routine to know what type of
argument is expected.

The va_end() function is used to clean up; it invalidates ap for use (unless va_start() is invoked
again).

Multiple traversals, each bracketed by va_start() ... va_end(), are possible.
EXAMPLE

This example is a possible implementation of `execl()`.

```c
#include <stdarg.h>

#define MAXARGS  31

/*
 *  execl is called by
 *  execl(file, arg1, arg2, ..., (char *)(0));
 */
int execl (const char *file, const char *args, ...)
{
    va_list ap;
    char *array[MAXARGS];
    int argno = 0;
    va_start(ap, args);
    while (args != 0) {
        array[argno++] = args;
        args = va_arg(ap, const char *);
    }
    va_end(ap);
    return execv(file, array);
}
```

APPLICATION USAGE

It is up to the calling routine to communicate to the called routine how many arguments there are, since it is not always possible for the called routine to determine this in any other way. For example, `execl()` is passed a null pointer to signal the end of the list. The `printf()` function can tell how many arguments are there by the `format` argument.

SEE ALSO

`exec`, `printf()`, `<varargs.h>`.

CHANGE HISTORY

First released in Issue 4.

Derived from the ANSI C standard.
NAME
stddef.h — standard type definitions

SYNOPSIS
#include <stddef.h>

DESCRIPTION
The <stddef.h> header defines the following:

 NULL Null pointer constant.

 offsetof(type, member-designator)
 Integral constant expression of type size_t, the value of which is the offset in bytes to the structure member (member-designator), from the beginning of its structure (type).

The <stddef.h> header defines through typedef:

 ptrdiff_t Signed integral type of the result of subtracting two pointers.

 wchar_t Integral type whose range of values can represent distinct wide character codes for all members of the largest character set specified among the locales supported by the compilation environment: the null character has the code value 0 and each member of the Portable Character Set has a code value equal to its value when used as the lone character in an integer character constant.

 size_t Unsigned integral type of the result of the sizeof operator.

SEE ALSO
<wchar.h>, <sys/types.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the ANSI C standard.
NAME
stdio.h — standard buffered input/output

SYNOPSIS
#include <stdio.h>

DESCRIPTION
The <stdio.h> header defines the following macro names as positive integral constant expressions:

BUFSIZ Size of <stdio.h> buffers.
FILENAME_MAX Maximum size in bytes of the longest filename string that the implementation guarantees can be opened.
FOPEN_MAX Number of streams which the implementation guarantees can be open simultaneously. The value will be at least eight.
_IOFBF Input/output fully buffered.
_IOLBF Input/output line buffered.
_IONBF Input/output unbuffered.
L_ctermid Maximum size of character array to hold ctermid() output
L_cuserid Maximum size of character array to hold cuserid() output (TO BE WITHDRAWN).
L_tmpnam Maximum size of character array to hold tmpnam() output.
SEEK_CUR Seek relative to current position.
SEEK_END Seek relative to end-of-file.
SEEK_SET Seek relative to start-of-file.
TMP_MAX Minimum number of unique filenames generated by tmpnam().
EX Maximum number of times an application can call tmpnam() reliably. The value of TMP_MAX will be at least 10,000.

The following macro name is defined as a negative integral constant expression:
EOF End-of-file return value.

The following macro name is defined as a null pointer constant:
NULL Null pointer.

The following macro name is defined as a string constant:
P_tmpdir default directory prefix for tempnam().

The following macro names are defined as expressions of type pointer to FILE:
stderr Standard error output stream.
stdin Standard input stream.
stdout Standard output stream.

The following data types are defined through typedef:
FILE A structure containing information about a file.
fpos_t Type containing all information needed to specify uniquely every position within a file.

va_list As described in <stdarg.h>.
size_t As described in <stddef.h>.
The following are declared as functions and may also be defined as macros:

```c
void clearerr(FILE *stream);
char *ctermid(char *s);

EX int fclose(FILE *stream);
FILE *fdopen(int fildes, const char *mode);
int feof(FILE *stream);
int ferror(FILE *stream);
int fflush(FILE *stream);
int fgetc(FILE *stream);
int fgetpos(FILE *stream, fpos_t *pos);
char *fgets(char *s, int n, FILE *stream);
int fileno(FILE *stream);
FILE *fopen(const char *filename, const char *mode);
int fprintf(FILE *stream, const char *format, ...);
int fputc(int c, FILE *stream);
int fputs(const char *s, FILE *stream);
size_t fread(void *ptr, size_t size, size_t nitems, FILE *stream);
FILE *freopen(const char *filename, const char *mode, FILE *stream);
int fscanf(FILE *stream, const char *format, ...);
int fseek(FILE *stream, long int offset, int whence);
int fsetpos(FILE *stream, const fpos_t *pos);
long int ftell(FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t nitems, FILE *stream);
int getc(FILE *stream);
int getchar(void);

EX int getopt(int argc, char *const argv[],
const char *optstring); (TO BE WITHDRAWN)
char *gets(char *s);

EX int getw(FILE *stream);
int pclose(FILE *stream);
void perror(const char *s);
FILE *popen(const char *command, const char *type);
int printf(const char *format, ...);
int putc(int c, FILE *stream);
int putchar(int c);
int puts(const char *s);

EX int putw(int w, FILE *stream);
int remove(const char *path);
int rename(const char *old, const char *new);
void rewind(FILE *stream);
int scanf(const char *format, ...);
void setbuf(FILE *stream, char *buf);
int setvbuf(FILE *stream, char *buf, int type, size_t size);
int sprintf(char *s, const char *format, ...);
int sscanf(const char *s, const char *format, int ...);

EX int tmpnam(const char *dir, const char *pfx);
FILE *tmpfile(void);
char *tmpnam(char *s);
int ungetc(int c, FILE *stream);
int vfprintf(FILE *stream, const char *format, va_list ap);
int vprintf(const char *format, va_list ap);
int vsprintf(char *s, const char *format, va_list ap);
```
The following external variables are defined:

```c
extern char *optarg;  
extern int opterr;  
extern int optind;  
extern int optopt;  
```

(In TO BE WITHDRAWN)

Inclusion of the `<stdio.h>` header may also make visible all symbols from `<stddef.h>`.

SEE ALSO

clearerr(), ctermid(), cuserid(), fclose(), fdopen(), fgetc(), fgetpos(), ferror(), feof(), fflush(), fsets(), fileno(), fopen(), fputc(), fputs(), fread(), freopen(), fseek(), fsetpos(), fsize(), getc(), getwchar(), gets(), getchar(), getopt(), gets(), getw(), pclose(), perror(), popen(), printf(), putc(), putchar(), puts(), putw(), putwchar(), remove(), rename(), rewind(), scanf(), setbuf(), setvbuf(), sscanf(), stdin, system(), tempnam(), tmpfile(), tmpnam(), ungetc(), vprintf(), `<sys/types.h>`.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated for alignment with the ISO C standard:

- The function declarations in this header are expanded to full ISO C prototypes.
- The DESCRIPTION section is restructured to group lists of macro names according to how they will be defined by an implementation (for example, whether they are integral constant expressions, pointer constants or string constants).
- The constant FILENAME_MAX is added to the list of integral constant expressions. The text of OPEN_MAX has also been changed for consistency with the ISO C standard.
- The data type fpos_t is moved from the APPLICATION USAGE section to the DESCRIPTION section.
- The functions fgetpos() and fsetpos() are added to the list of functions declared in this header.

Other changes are incorporated as follows:

- The constant L_cuserid and the external variables optarg, opterr, optind and optopt are marked as extensions and TO BE WITHDRAWN.
- The minimum allowable value of TMP_MAX, 10,000 on XSI-conformant systems, has been marked as an extension.
- The P_tmpdir constant is moved from the APPLICATION USAGE section to the DESCRIPTION section and marked as an extension. The remainder of the APPLICATION USAGE section is removed.
- References to the va_list and size_t types are added to the DESCRIPTION section.
- Function declarations of the cuserid(), getopt(), getw(), putw() and tempnam() functions, and the va_list type are marked as extensions.
- The cuserid() and getopt() functions are marked TO BE WITHDRAWN.
- A warning is added indicating that inclusion of `<stdio.h>` may also make visible all symbols from `<stddef.h>`.
NAME
stdlib.h — standard library definitions

SYNOPSIS
#include <stdlib.h>

DESCRIPTION
The <stdlib.h> header defines the following macro names:

- EXIT_FAILURE: Unsuccessful termination for exit(), evaluates to a non-zero value.
- EXIT_SUCCESS: Successful termination for exit(), evaluates to 0.
- NULL: Null pointer.
- RAND_MAX: Maximum value returned by rand(), at least 32,767.
- MB_CUR_MAX: Integer expression whose value is the maximum number of bytes in a character specified by the current locale.

The following data types are defined through typedef:

- div_t: Structure type returned by div() function.
- ldiv_t: Structure type returned by ldiv() function.
- size_t: As described in <stddef.h>.
- wchar_t: As described in <stddef.h>.

In addition, the following symbolic names and macros are defined as in <sys/wait.h>, for use in decoding the return value from system():

- WNOHANG
- WUNTRACED
- WEXITSTATUS()
- WIFEXITED()
- WIFSIGNALED()
- WIFSTOPPED()
- WSTOPSIG()
- WTERMSIG()

The following are declared as functions and may also be defined as macros:

- long a64l(const char *s);
- void abort(void);
- int abs(int i);
- int atexit(void (*func)(void));
- double atof(const char *str);
- int atoi(const char *str);
- long int atol(const char *str);
- void *bsearch(const void *key, const void *base, size_t nel, size_t width, int (*compar)(const void *, const void *));
- void *calloc(size_t nelem, size_t elsize);
- div_t div(int numer, int denom);
- double drand48(void);
- double erand48(unsigned short int xsubi[3]);
- char *ecvt (double value, int ndigit, int *decpt, int *sign);
<stdlib.h>

void exit(int status);

char *fcvt(double value, int ndigit, int *decpt, int *sign);
void free(void *ptr);

char *gcvt(double value, int ndigit, char *buf);
char *getenv(const char *name);

int getsubopt(char **optionp, char *const *tokens, char **valuep);
int granpt(int fildes);
char *initstate(unsigned seed, char *state, int size);

long int jrand48(unsigned short int xsubi[3]);
char *l64a(long value);

long int labs(long int j);

void lcong48(unsigned short int param[7]);
long int lrand48(void);
void *malloc(size_t size);
int mblen(const char *s, size_t n);
size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n);
int mbtowc(wchar_t *pwc, const char *s, size_t n);

char *mktemp(char *template);
char *mkstemp(char *template);
long int mrand48(void);
long int nrand48(unsigned short int xsubi[3]);
char *ptsname(int fildes);
int putenv(const char *string);

void qsort(void *base, size_t nel, size_t width,
    int (*compar)(const void *, const void *));
int rand(void);

long int random(void);
void realloc(void *ptr, size_t size);

char *realpath(const char *file_name, char *resolved_name);
unsigned short int *seed48(unsigned short int seed16v[3]);
void setkey(const char *key);
char *setstate(char *state);
void srand(unsigned int seed);

void *valloc(size_t size); (TO BE WITHDRAWN)

double strtod(const char *str, char **ptr);
long int strtol(const char *str, char **ptr, int base);
unsigned long int strtoul(const char *str, char **ptr, int base);
int system(const char *string);

int ttyslot(void); (TO BE WITHDRAWN)
int unlockpt(int fildes);
void *valloc(size_t size); (TO BE WITHDRAWN)
size_t wcstombs(char *s, const wchar_t *pwcs, size_t n);
int wcwtomb(char *s, wchar_t wchar);

SEE ALSO
    a64l(), abort(), abs(), atexit(), atof(), atoi(), atol(), bsearch(), calloc(), div(), drand48(), ecvt(), erand48(), exit(), fcntl(), free(), gcvt(), getenv(), getsubopt(), grantpt(), initstate(), jrand48(), l64a(), labs(), lcong48(), ldiv(), lrand48(), malloc(), mblen(), mbstowcs(), mbtowc(), mktemp(), mkstemp(),
Headers

<stdlib.h>

mrand48(), nrand48(), ptsname(), putenv(), qsort(), rand(), realloc(), realpath(), setstate(), srand(), srand48(), srand48(), rand48(), strtd() , strtol(), srandom(), strtoul(), ttyslot(), unlockpt(), valloc(), wcstombs(), wctomb(), <sys/types.h>.

CHANGE HISTORY
First released in Issue 3.

Issue 4
The following changes are incorporated for alignment with the ISO C standard:

- The function declarations in this header are expanded to full ISO C prototypes.
- The maximum value of RAND_MAX is defined.
- The name MB_CUR_MAX is added to the list of macro names defined in this header, while div_t and ldiv_t are added to the list of defined types.
- The names atexit(), div(), labs(), ldiv(), mblen(), mbstowcs(), mbtowc(), strtoul(), wcstombs() and wctomb() are added to the list of functions declared in this header.

Other changes are incorporated as follows:

- A reference is added to <stddef.h> and <wchar.h> for the definition of size_t.
- A reference is added to <sys/wait.h> for definitions of the symbolic names and macros defined for decoding the return value from the system() function. This reference and the symbolic names and macros are marked as an extension.
- The names drand48(), erand48(), jrand48(), lcong48(), lrand48(), mrand48(), nrand48(), putenv(), seed48(), setkey() and srand48() are added to the list of functions declared in this header and marked as extensions.
- A warning is added indicating that inclusion of <stdlib.h> may also make visible all symbols from <stddef.h>, <limits.h>, <math.h> and <sys/wait.h>.
- The APPLICATION USAGE section is removed.

Issue 4, Version 2
For X/OPEN UNIX conformance, the a64l(), ecvt(), fcvt(), gcvt(), getsubopt(), grantpt(), initstate(), l64a(), mktemp(), mkstemp(), ptsname(), random(), realpath(), setstate(), srandom(), ttyslot(), unlockpt() and valloc() functions are added to the list of functions declared in this header.
NAME

string.h — string operations

SYNOPSIS

#include <string.h>

DESCRIPTION

The <string.h> header defines the following:

NULL   Null pointer constant.

size_t  As described in <stddef.h>.

The following are declared as functions and may also be defined as macros:

EX
void *memccpy(void * s1 , const void * s2 , int c , size_t n);
void *memchr(const void * s , int c , size_t n);
int memcmp(const void * s1 , const void * s2 , size_t n);
void *memcpy(void * s1 , const void * s2 , size_t n);
void *memmove(void * s1 , const void * s2 , size_t n);
void *memset(void * s , int c , size_t n);
char *strcat(char * s1 , const char * s2);
char *strchr(const char * s , int c);
int strcmp(const char * s1 , const char * s2);
size_t strlen(const char * s);
char *strdup(const char * s1);
char *strerror(int errnum);
size_t strcspn(const char * s1 , const char * s2);
char *strchr(const char * s , int c);
int strncmp(const char * s1 , const char * s2 , size_t n);
char *strncpy(char * s1 , const char * s2 , size_t n);
char *strpbrk(const char * s1 , const char * s2);
char *strrchr(const char * s , int c);
int strcoll(const char * s1 , const char * s2);
size_t strcspn(const char * s1 , const char * s2);
char *strchr(const char * s , int c);
size_t strcoll(const char * s1 , const char * s2);

UX
char *strdup(const char * s1);
char *strerror(int errnum);
size_t strlen(const char * s);
char *strncat(char * s1 , const char * s2 , size_t n);
int strncmp(const char * s1 , const char * s2 , size_t n);
char *strncpy(char * s1 , const char * s2 , size_t n);
char *strpbrk(const char * s1 , const char * s2);
char *strrchr(const char * s , int c);
size_t strcspn(const char * s1 , const char * s2);
char *strstr(const char * s1 , const char * s2);
char *strtok(char * s1 , const char * s2);
size_t strxfrm(char * s1 , const char * s2 , size_t n);

EX

Inclusion of the <string.h> header may also make visible all symbols from <stddef.h>.

SEE ALSO

memccpy(), memchr(), memcmp(), memcpy(), memmove(), memset(), strcat(), strchr(), strcmp(),
strcoll(), strcspn(), strcspn(), strlen(), strerror(), strlend(), strncat(), strncoll(), strncpy(),
strncpy(), strpbrk(), strstr(), strspn(), strtok(), strxfrm(), <sys/types.h>.

CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated for alignment with the ISO C standard:

• The function declarations in this header are expanded to full ISO C prototypes.

• The name memmove() is added to the list of functions declared in this header.
Other changes are incorporated as follows:

- A reference is added to `<stddef.h>` for the definition of `size_t`.
- The `memccpy()` function is marked as an extension.
- A warning is added indicating that inclusion of `<string.h>` may also make visible all symbols from `<stddef.h>`.
- The APPLICATION USAGE section is removed.

**Issue 4, Version 2**

For X/OPEN UNIX conformance, the `strdup()` function is added to the list of functions declared in this header.
NAME

strings — string operations

SYNOPSIS

UX

#include <strings.h>

DESCRIPTION

The following are declared as functions and may also be defined as macros:

    int bcmp(const void *s1, const void *s2, size_t n);
    void bcopy(const void *s1, void *s2, size_t n);
    void bzero(void *s, size_t n);
    int ffs(int i);
    char *index(const char *s, int c);
    char *rindex(const char *s, int c);
    int strcasecmp(const char *s1, const char *s2);
    int strncasecmp(const char *s1, const char *s2, size_t n);

SEE ALSO

    bcmp(), bcopy(), bzero(), ffs(), index(), rindex(), strcasecmp().

CHANGE HISTORY

    First released in Issue 4, Version 2.
NAME
stropts.h — STREAMS interface

SYNOPSIS
UX
#include <stropts.h>

DESCRIPTION
The <stropts.h> header defines the bandinfo structure that includes at least the following members:

unsigned char bi_pri
int bi_flag

The <stropts.h> header defines the strpeek structure that includes at least the following members:

struct strbuf ctlbuf
struct strbuf databuf
long flags

The <stropts.h> header defines the strbuf structure that includes at least the following members:

int maxlen maximum buffer length
int len length of data
char *buf ptr to buffer

The <stropts.h> header defines the strfdinsert structure that includes at least the following members:

struct strbuf ctlbuf
struct strbuf databuf
long flags
int fildes
int offset

The <stropts.h> header defines the strioctl structure that includes at least the following members:

int ic_cmd
int ic_timout
int ic_len
char *ic_dp

The <stropts.h> header defines the strrecvfd structure that includes at least the following members:

int fd
uid_t uid
gid_t gid

The <stropts.h> header defines the str_list structure that includes at least the following members:

int sl_nmods
struct str_mlist *sl_modlist

The <stropts.h> header defines the str_mlist structure that includes at least the following member:

char l_name[FMNAMESZ+1]
At least the following macros are defined for use as the request argument to ioctl():

I_PUSH Push STREAMS module onto the top of the current STREAM, just below the STREAM head.

I_POP Remove STREAMS module from just below the STREAM head.

I_LOOK Retrieve the name of the module just below the STREAM head and place it in a character string. At least the following macros are defined for use as the arg argument:

FMNAMESZ The minimum size in bytes of the buffer referred to by the arg argument.

I_FLUSH This request flushes all input and/or output queues, depending on the value of the arg argument. At least the following macros are defined for use as the arg argument:

FLUSHR Flush read queues.
FLUSHW Flush write queues.
FLUSHRW Flush read and write queues.

I_FLUSHBAND Flush only band specified.

I_SETSIG Informs the STREAM head that the process wants the SIGPOLL signal issued (see signal() and sigset()) when a particular event has occurred on the STREAM.

The header <stropts.h> defines these possible values for arg when I_SETSIG is specified:

S_RDNORM A normal (priority band set to 0) message has arrived at the head of a STREAM head read queue.

S_RDBAND A message with a non-zero priority band has arrived at the head of a STREAM head read queue.

S_INPUT A message, other than a high-priority message, has arrived at the head of a STREAM head read queue.

S_HIPRI A high-priority message is present on a STREAM head read queue.

S_OUTPUT The write queue for normal data (priority band 0) just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) normal data downstream.

S_WRNORM Same as S_OUTPUT.

S_WRBAND The write queue for a non-zero priority band just below the STREAM head is no longer full.

S_MSG A STREAMS signal message that contains the SIGPOLL signal reaches the front of the STREAM head read queue.

S_ERROR Notification of an error condition reaches the STREAM head.

S_HANGUP Notification of a hangup reaches the STREAM head.
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.

I_GETSIG Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal.

I_FIND Compares the names of all modules currently present in the STREAM to the name pointed to by arg.

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 arg argument:

RS_HIPRI Only look for high-priority messages.

I_SRDOPT Sets the read mode. At least the following macros are defined for use as the arg argument:

RNORM Byte-STREAM mode, the default.
RMSGD Message-discard mode.
RMSGN Message-nondiscard mode.
RPROTNORM Fail read() with [EBADMSG] if a message containing a control part is at the front of the STREAM head read queue.
RPROTDAT Deliver the control part of a message as data when a process issues a read().
RPROTDIS Discard the control part of a message, delivering any data part, when a process issues a read().

I_GRDOPT Returns the current read mode setting.

I_NREAD Counts the number of data bytes in data blocks in the first message on the STREAM head read queue.

I_FDINSERT Creates a message from the specified buffer(s), adds information about another STREAM, and sends the message downstream.

I_STR Constructs an internal STREAMS ioctl() message and sends that message downstream.

I_SWROPT Sets the write mode. At least the following macros are defined for use as the arg argument:

SNDZERO Send a zero-length message downstream when a write() of 0 bytes occurs.

I_GWROPT Returns the current write mode setting.

I_SENDFD Requests the STREAM associated with fd to send a message, containing a file pointer, to the STREAM head at the other end of a STREAMS pipe.

I_RECVFD Retrieves the file descriptor associated with the message sent by an I_SENDFD ioctl() over a STREAMS pipe.

I_LIST This request allows the process to list all the module names on the STREAM, up to and including the topmost driver name.
This request allows the process to see if the current message on the STREAM head read queue is "marked" by some module downstream. At least the following macros are defined for use as the arg argument:

- **ANYMARK**: Check if the message is marked.
- **LASTMARK**: Check if the message is the last one marked on the queue.

I_CKBAND: Check if the message of a given priority band exists on the STREAM head read queue.

I_GETBAND: Return the priority band of the first message on the STREAM head read queue.

I_CANPUT: Check if a certain band is writable.

I_SETCLTIME: Allows the process to set the time the STREAM head will delay when a STREAM is closing and there is data on the write queues.

I_GETCLTIME: Returns the close time delay.

I_LINK: Connects two STREAMs.

I_UNLINK: Disconnects the two STREAMs. The header defines at least the following value for all:

- **MUXID_ALL**: Unlink all STREAMs linked to the STREAM associated with fd.

I_PLINK: Connects two STREAMs with a persistent link.

I_PUNLINK: Disconnects the two STREAMs that were connected with a persistent link.

The following macros are defined for `getmsg()`, `getpmsg()`, `putmsg()` and `putpmsg()`:

- **MSG_ANY**: Receive any message.
- **MSG_BAND**: Receive message from specified band.
- **MSG_HIPRI**: Send/Receive high priority message.
- **MORECTL**: More control information is left in message.
- **MOREDATA**: More data is left in message.

The header `<stropts.h>` may make visible all of the symbols from `<unistd.h>`.

The following are declared as functions in the `<stropts.h>` header and may also be defined as macros:

```c
int isastream(int fildes);
int getmsg(int fd, struct strbuf *ctlptr, struct strbuf *dataptr, int *flagsp);
int getpmsg(int fd, struct strbuf *ctlptr, struct strbuf *dataptr, int *bandp, int *flagsp);
int ioctl(int fildes, int request, ...);
int putmsg(int fd, const struct strbuf *ctlptr, const struct strbuf *dataptr, int flags);
int putpmsg(int fd, const struct strbuf *ctlptr, const struct strbuf *dataptr, int band, int flags);
int fattach(int fildes, const char *path);
int fdetach(const char *path);
```
SEE ALSO
   close(), fcntl(), getmsg(), ioctl(), open(), pipe(), read(), poll(), putmsg(), signal(), sigset(), write().

CHANGE HISTORY
   First released in Issue 4, Version 2.
NAME
syslog — definitions for system error logging

SYNOPSIS
UX
#include <syslog.h>

DESCRIPTION
The <syslog.h> header defines the following symbolic constants, zero or more of which may be
OR-ed together to form the logopt option of openlog():

LOG_PID Log the process ID with each message.
LOG_CONS Log to the system console on error.
LOG_NDELAY Connect to syslog daemon immediately.
LOG_ODELAY Delay open until syslog() is called.
LOG_NOWAIT Don’t wait for child processes.

The following symbolic constants are defined as possible values of the facility argument to
openlog():

LOG_KERN Reserved for message generated by the system.
LOG_USER Message generated by a process.
LOG_MAIL Reserved for message generated by mail system.
LOG_NEWS Reserved for message generated by news system.
LOG_UUCP Reserved for message generated by UUCP system.
LOG_DAEMON Reserved for message generated by system daemon.
LOG_AUTH Reserved for message generated by authorisation daemon.
LOG_CRON Reserved for message generated by the clock daemon.
LOG_LPR Reserved for message generated by printer system.
LOG_LOCAL0 Reserved for local use.
LOG_LOCAL1 Reserved for local use.
LOG_LOCAL2 Reserved for local use.
LOG_LOCAL3 Reserved for local use.
LOG_LOCAL4 Reserved for local use.
LOG_LOCAL5 Reserved for local use.
LOG_LOCAL6 Reserved for local use.
LOG_LOCAL7 Reserved for local use.

The following are declared as macros for constructing the maskpri argument to setlogmask(). The
following macros expand to an expression of type int when the argument pri is an expression of
type int:

LOG_MASK(pri) A mask for priority pri.
LOG_UPTO(pri) A mask for all priorities through pri.

The following constants are defined as possible values for the priority argument of syslog():

LOG_EMERG A panic condition was reported to all processes.
LOG_ALERT A condition that should be corrected immediately.
LOG_CRIT A critical condition.
LOG_ERR An error message.
LOG_WARNING A warning message.
LOG_NOTICE A condition requiring special handling.
LOG_INFO A general information message.
LOG_DEBUG A message useful for debugging programs.
The following are declared as functions and may also be defined as macros:

```c
void closelog(void);
void openlog(const char *id, int logopt, int facility);
int setlogmask(int maskpri);
void syslog(int priority, const char *format, ...);
```

SEE ALSO

`closelog()`.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
sys/ipc.h — interprocess communication access structure

SYNOPSIS
#include <sys/ipc.h>

DESCRIPTION
The <sys/ipc.h> header is used by three mechanisms for interprocess communication (IPC): messages, semaphores and shared memory. All use a common structure type, \texttt{ipc_perm} to pass information used in determining permission to perform an IPC operation.

The structure \texttt{ipc_perm} contains the following members:

\begin{verbatim}
uid_t   uid    owner's user ID
gid_t   gid    owner's group ID
uid_t   cuid   creator's user ID
gid_t   cgid   creator's group ID
mode_t  mode   read/write permission
\end{verbatim}

The \texttt{uid_t}, \texttt{gid_t}, \texttt{mode_t} and \texttt{key_t} types are defined as described in <sys/types.h>.

Definitions are given for the following constants:

Mode bits:
\begin{verbatim}
IPC_CREAT Create entry if key does not exist.
IPC_EXCL Fail if key exists.
IPC_NOWAIT Error if request must wait.
\end{verbatim}

Keys:
\begin{verbatim}
IPC_PRIVATE Private key.
\end{verbatim}

Control Commands:
\begin{verbatim}
IPC_RMID Remove identifier.
IPC_SET Set options.
IPC_STAT Get options.
\end{verbatim}

UX The following is declared as a function and may also be defined as a macro:

\begin{verbatim}
key_t ftok(const char *path, int id);
\end{verbatim}

SEE ALSO
ftok(), <sys/types.h>.

CHANGE HISTORY
First released in Issue 2.
Derived from System V Release 2.0.
Issue 4
The following changes are incorporated in this issue:

- The DESCRIPTION is corrected to say that the header “is used by three mechanisms . . .”.
- Reference to the header <sys/types.h> is added for the definitions of uid_t, gid_t and mode_t.

Issue 4, Version 2
For X/OPEN UNIX conformance, the ftok() function is added to the list of functions declared in this header.
NAME
sys/mman.h — memory management declarations

SYNOPSIS
UX

#include <sys/mman.h>

DESCRIPTION
The following protection options are defined:

- PROT_READ  Page can be read.
- PROT_WRITE Page can be written.
- PROT_EXEC  Page can be executed.
- PROT_NONE Page can not be accessed.

The following flag options are defined:

- MAP_SHARED Share changes.
- MAP_PRIVATE Changes are private.
- MAP_FIXED  Interpret addr exactly.

The following flags are defined for msync():

- MS_ASYNC  Perform asynchronous writes.
- MS_SYNC   Perform synchronous writes.
- MS_INVALIDATE   Invalidate mappings.

The size_t and off_t types are defined as described in <sys/types.h>.

The following are declared in <sys/mman.h> as functions and may also be defined as macros:

- void *mmap(void * addr, size_t len, int prot, int flags, int fd, off_t off);
- int mprotect(void * addr, size_t len, int prot);
- int msync(void * addr, size_t len, int flags);
- int munmap(void * addr, size_t len);

SEE ALSO
mmap(), mprotect(), msync(), munmap().

CHANGE HISTORY
First released in Issue 4, Version 2.
NAME
sys/msg.h — message queue structures

SYNOPSIS
#include <sys/msg.h>

DESCRIPTION
The <sys/msg.h> header defines the following constant and members of the structure msqid_ds.

The following data types are defined through typedef:

msgqnum_t Used for the number of messages in the message queue.
msglen_t Used for the number of bytes allowed in a message queue.

These types are unsigned integer types that are able to store values at least as large as a type unsigned short.

Message operation flag:
MSG_NOERROR No error if big message.

The structure msqid_ds contains the following members:

struct ipc_perm msg_perm operation permission structure
msgqnum_t msg_qnum number of messages currently on queue
msglen_t msg_qbytes maximum number of bytes allowed on queue
pid_t msg_lspid process ID of last msgsnd()
pid_t msg_lrpid process ID of last msgrcv()
time_t msg_stime time of last msgsnd()
time_t msg_rtime time of last msgrcv()
time_t msg_ctime time of last change

The pid_t, time_t, key_t and size_t types are defined as described in <sys/types.h>.

The following are declared as functions and may also be defined as macros:

int msgctl(int msqid, int cmd, struct msqid_ds *buf);
int msgget(key_t key, int msgflg);
int msgrcv(int msqid, void *msgp, size_t msgsz, long int msgtyp,
int msgflg);
int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);

In addition, all of the symbols from <sys/ipc.h> will be defined when this header is included.

SEE ALSO
msgctl(), msgget(), msgrcv(), msgsnd(), <sys/types.h>.

CHANGE HISTORY
First released in Issue 2.
Derived from System V Release 2.0.
Issue 4

The following changes are incorporated in this issue:

- The function declarations in this header are expanded to full ISO C prototypes.
- Reference to the header `<sys/types.h>` is added for the definitions of `pid_t`, `time_t`, `key_t` and `size_t`.
- A statement is added indicating that all symbols in `<sys/ipc.h>` will be defined when this header is included.
NAME
sys/resource.h — definitions for XSI resource operations

SYNOPSIS
UX
#include <sys/resource.h>

DESCRIPTION
The <sys/resource.h> header defines the following symbolic constants as possible values of the which argument of getpriority() and setpriority():

PRIO_PROCESS Identifies who argument as a process ID.
PRIO_PGRP Identifies who argument as a process group ID.
PRIO_USER Identifies who argument as a user ID.

The following type is defined through typedef:

rlim_t Unsigned integral type used for limit values.

The following symbolic constant is defined:

RLIM_INFINITY A value of rlim_t indicating no limit.

The following symbolic constants are defined as possible values of the who parameter of getrusage():

RUSAGE_SELF Returns information about the current process.
RUSAGE_CHILDREN Returns information about children of the current process.

The <sys/resource.h> header defines the rlimit structure that includes at least the following members:

rlim_t rlim_cur the current (soft) limit
rlim_t rlim_max the hard limit

The <sys/resource.h> header defines the rusage structure that includes at least the following members:

struct timeval ru_utime user time used
struct timeval ru_stime system time used

The timeval structure is defined as described in <sys/time.h>.

The following symbolic constants are defined as possible values for the resource argument of getrlimit() and setrlimit():

RLIMIT_CORE Limit on size of core dump file.
RLIMIT_CPU Limit on CPU time per process.
RLIMIT_DATA Limit on data segment size.
RLIMIT_FSIZE Limit on file size.
RLIMIT_NOFILE Limit on number of open files.
RLIMIT_STACK Limit on stack size.
RLIMIT_AS Limit on address space size.

The following are declared as functions and may also be defined as macros:

int getpriority(int which, id_t who);
int getrlimit(int resource, struct rlimit *rlp);
int getrusage(int who, struct rusage *r_usage);
int setpriority(int which, id_t who, int priority);
int setrlimit(int resource, const struct rlimit *rlp);
Inclusion of the `<sys/resource.h>` header may also make visible all symbols from `<sys/time.h>`.

SEE ALSO

`getpriority()`, `getusage()`, `getrlimit()`.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
sys/sem.h — semaphore facility

SYNOPSIS
EX
#include <sys/sem.h>

DESCRIPTION
The <sys/sem.h> header defines the following constants and structures.

Semaphore operation flags:
SEM_UNDO Set up adjust on exit entry.

Command definitions for the function semctl():
GETNCNT Get semncnt.
GETPID Get sempid.
GETVAL Get semval.
GETALL Get all cases of semval.
GETZCNT Get semzcnt.
SETVAL Set semval.
SETALL Set all cases of semval.

The structure semid_ds contains the following members:

struct ipc_perm sem_perm operation permission structure
unsigned short int sem_nsems number of semaphores in set
time_t sem_otime last semop() time
time_t sem_ctime last time changed by semctl()

The pid_t, time_t, key_t and size_t types are defined as described in <sys/types.h>.

A semaphore is represented by an anonymous structure containing the following members:

unsigned short int semval semaphore value
pid_t sempid process ID of last operation
unsigned short int semncnt number of processes waiting for semval to become greater than current value
unsigned short int semzcnt number of processes waiting for semval to become 0

The structure sembuf contains the following members:

unsigned short int sem_num semaphore number
short int sem_op semaphore operation
short int sem_flg operation flags

The following are declared as functions and may also be defined as macros:

int semctl(int semid, int semnum, int cmd, ...);
int semget(key_t key, int nsems, int semflg);
int semop(int semid, struct sembuf *sops, size_t nsops);

In addition, all of the symbols from <sys/ipc.h> will be defined when this header is included.

SEE ALSO
semctl(), semget(), semop(), <sys/types.h>.
CHANGE HISTORY
First released in Issue 2.
Derived from System V Release 2.0.

Issue 4
The following changes are incorporated in this issue:

- The function declarations in this header are expanded to full ISO C prototypes.
- Reference to the header `<sys/types.h>` is added for the definitions of `pid_t`, `time_t`, `key_t` and `size_t`.
- A statement is added indicating that all symbols in `<sys/ipc.h>` will be defined when this header is included.
NAME

sys/shm.h — shared memory facility

SYNOPSIS

```c
#include <sys/shm.h>
```

DESCRIPTION

The `<sys/shm.h>` header defines the following symbolic constants and structure:

Symbolic constants:

- `SHM_RDONLY` Attach read-only (else read-write).
- `SHMLBA` Segment low boundary address multiple.
- `SHM_RND` Round attach address to SHMLBA.

The following data types are defined through `typedef`:

- `shmatt_t` Signed integer used for the number of current attaches that must be able to store values at least as large as a type `unsigned short`.

The structure `shmid_ds` contains the following members:

```c
define struct ipc_perm
  shm_perm          operation permission structure
int                shm_segsz    size of segment in bytes
pid_t              shm_lpid     process ID of last shared memory operation
pid_t              shm_cpid     process ID of creator
shmatt_t           shm_nattch   number of current attaches
time_t             shm_atime    time of last `shmat()`
time_t             shm_dtime    time of last `shmdt()`
time_t             shm_ctime    time of last change by `shmctl()`
```

The `pid_t`, `time_t`, `key_t` and `size_t` types are defined as described in `<sys/types.h>`. The following are declared as functions and may also be defined as macros:

```c
void *shmat(int shmid, const void *shmaddr, int shmflg);
int   shmctl(int shmid, int cmd, struct shmid_ds *buf);
int   shmdt(const void *shmaddr);
int   shmget(key_t key, size_t size, int shmflg);
```

In addition, all of the symbols from `<sys/ipc.h>` will be defined when this header is included.

SEE ALSO

`shmat()`, `shmctl()`, `shmdt()`, `shmget()`, `<sys/types.h>`.

CHANGE HISTORY

First released in Issue 2.

Derived from System V Release 2.0.

Issue 4

The following changes are incorporated in this issue:

- The function declarations in this header are expanded to full ISO C prototypes.
- Reference to the header `<sys/types.h>` is added for the definitions of `pid_t`, `time_t`, `key_t` and `size_t`.
- A statement is added indicating that all symbols in `<sys/ipc.h>` will be defined when this header is included.
NAME
sys/stat.h — data returned by the stat() function

SYNOPSIS
#include <sys/stat.h>

DESCRIPTION
The <sys/stat.h> header defines the structure of the data returned by the functions fstat(), lstat(), and stat().

The structure stat contains at least the following members:

- dev_t st_dev: ID of device containing file
- ino_t st_ino: file serial number
- mode_t st_mode: mode of file (see below)
- nlink_t st_nlink: number of links to the file
- uid_t st_uid: user ID of file
- gid_t st_gid: group ID of file
- dev_t st_rdev: device ID (if file is character or block special)
- off_t st_size: file size in bytes (if file is a regular file)
- time_t st_atime: time of last access
- time_t st_mtime: time of last data modification
- time_t st_ctime: time of last status change
- long st_blksize: a filesystem-specific preferred I/O block size for this object. In some filesystem types, this may vary from file to file.
- long st_blocks: number of blocks of a filesystem-specific size allocated for this object

File serial number and device ID taken together uniquely identify the file within the system. The dev_t, ino_t, mode_t, nlink_t, uid_t, gid_t, off_t and time_t types are defined as described in <sys/types.h>. Times are given in seconds since the Epoch.

The following symbolic names for the values of st_mode are also defined:

File type:
- S_IFMT: type of file
  - S_IFBLK: block special
  - S_IFCHR: character special
  - S_IFIFO: FIFO special
  - S_IFREG: regular
  - S_IFDIR: directory
- S_IFLNK: symbolic link
File mode bits:

- **S_IRWXU**: read, write, execute/search by owner
- **S_IRUSR**: read permission, owner
- **S_IWUSR**: write permission, owner
- **S_IXUSR**: execute/search permission, owner
- **S_IRWXG**: read, write, execute/search by group
- **S_IGRP**: read permission, group
- **S_IWGRP**: write permission, group
- **S_IXGRP**: execute/search permission, group
- **S_IRWXO**: read, write, execute/search by others
- **S_IROTH**: read permission, others
- **S_IWOTH**: write permission, others
- **S_IXOTH**: execute/search permission, others
- **S_ISUID**: set-user-ID on execution
- **S_ISGID**: set-group-ID on execution
- **S_ISVTX**: on directories, restricted deletion flag

The bits defined by S_IRUSR, S_IWUSR, S_IXUSR, S_IGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH, S_ISUID, S_ISGID and S_ISVTX are unique.

- **S_IRWXU** is the bitwise OR of S_IRUSR, S_IWUSR and S_IXUSR.
- **S_IRWXG** is the bitwise OR of S_IGRP, S_IWGRP and S_IXGRP.
- **S_IRWXO** is the bitwise OR of S_IROTH, S_IWOTH and S_IXOTH.

Implementations may OR other implementation-dependent bits into S_IRWXU, S_IRWXG and S_IRWXO, but they will not overlap any of the other bits defined in this document. The file permission bits are defined to be those corresponding to the bitwise inclusive OR of S_IRWXU, S_IRWXG and S_IRWXO.

The following macros will test whether a file is of the specified type. The value \( m \) supplied to the macros is the value of `st_mode` from a `stat` structure. The macro evaluates to a non-zero value if the test is true, 0 if the test is false.

- **S_ISBLK \((m)\)**: Test for a block special file.
- **S_ISCHR \((m)\)**: Test for a character special file.
- **S_ISDIR \((m)\)**: Test for a directory.
- **S_ISFIFO \((m)\)**: Test for a pipe or FIFO special file.
- **S_ISREG \((m)\)**: Test for a regular file.
- **S_ISLNK \((m)\)**: Test for a symbolic link.

The following are declared as functions and may also be defined as macros:

```c
int chmod(const char *path, mode_t mode);
int fchmod(int fildes, mode_t mode);
int fstat(int fildes, struct stat *buf);
int lstat(const char *path, struct stat *buf);
int mkdir(const char *path, mode_t mode);
int mkfifo(const char *path, mode_t mode);
int mknod(const char *path, mode_t mode, dev_t dev);
int stat(const char *path, struct stat *buf);
mode_t umask(mode_t cmask);
```

**APPLICATION USAGE**

Use of the macros is recommended for determining the type of a file.
SEE ALSO
   chmod(), fchmod(), fstat(), lstat(), mkdir(), mkfifo(), mknod(), stat(), umask(), <sys/types.h>.

CHANGE HISTORY
   First released in Issue 1.
   Derived from Issue 1 of the SVID.

Issue 4
   The following changes are incorporated for alignment with the ISO POSIX-1 standard:
   • The function declarations in this header are expanded to full ISO C prototypes.
   • The DESCRIPTION section is expanded to indicate (a) how files are uniquely identified
     within the system, (b) that times are given in units of seconds since the Epoch, (c) rules
     governing the definition and use of the file mode bits, and (d) usage of the file type test
     macros.

   Other changes are incorporated as follows:
   • Reference to the header <sys/types.h> is added for the definitions of dev_t, ino_t, mode_t,
     nlink_t, uid_t, gid_t, off_t and time_t. This has been marked as an extension.
   • References to the S_IREAD, S_IWRITE, S_IEXEC file and S_ISVTX modes are removed.
   • The descriptions of the members of the stat structure in the DESCRIPTION section are
     corrected.

Issue 4, Version 2
   The following changes are incorporated for X/OPEN UNIX conformance:
   • The st_blksize and st_blocks members are added to the stat structure.
   • The S_IFLINK value of S_IFMT is defined.
   • The S_ISVTX file mode bit and the S_ISLNK file type test macro is defined.
   • The fchmod(), lstat() and mknod() functions are added to the list of functions declared in this
     header.
NAME

sys/statvfs.h — VFS Filesystem information structure

SYNOPSIS

UX

```c
#include <sys/statvfs.h>
```

DESCRIPTION

The `<sys/statvfs.h>` header defines the `statvfs` structure that includes at least the following members:

- `unsigned long f_bsize` — file system block size
- `unsigned long f_frsize` — fundamental filesystem block size
- `unsigned long f_blocks` — total number of blocks on file system in units of `f_frsize`
- `unsigned long f_bfree` — total number of free blocks
- `unsigned long f_bavail` — number of free blocks available to non-privileged process
- `unsigned long f_files` — total number of file serial numbers
- `unsigned long f_ffree` — total number of free file serial numbers
- `unsigned long f_favail` — number of file serial numbers available to non-privileged process
- `unsigned long f_fsid` — file system id
- `unsigned long f_flag` — bit mask of `f_flag` values
- `unsigned long f_namemax` — maximum file length

The following flags for the `f_flag` member are defined:

- `ST_RDONLY` — read-only file system
- `ST_NOSUID` — does not support setuid/setgid semantics

The header `<sys/statvfs.h>` declares the following functions which may also be defined as macros:

- `int statvfs(const char *path, struct statvfs *buf);`
- `int fstatvfs(int fildes, struct statvfs *buf);`

SEE ALSO

`fstatvfs()`, `statvfs()`.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
sys/time.h — time types

SYNOPSIS
UX
#include <sys/time.h>

DESCRIPTION
The `<sys/time.h>` header defines the `timeval` structure that includes at least the following members:

```c
  time_t tv_sec seconds since Jan. 1, 1970
  long tv_usec microseconds
```

The `<sys/time.h>` header defines the `itimerval` structure that includes at least the following members:

```c
  struct timeval it_interval timer interval
  struct timeval it_value current value
```

The `time_t` type is defined as described in `<sys/types.h>`.

The `<sys/time.h>` header defines the `fd_set` type as a structure that includes at least the following member:

```c
  long fds_bits[] bit mask for open file descriptions
```

The `<sys/time.h>` header defines the following values for the `which` argument of `getitimer()` and `setitimer()`:

- `ITIMER_REAL` Decrement in real time.
- `ITIMER_VIRTUAL` Decrement in process virtual time.
- `ITIMER_PROF` Decrement both in process virtual time and when the system is running on behalf of the process.

The following macros are defined:

```c
void FD_CLR(int fd, fd_set * fdset)
        Clears the bit for the file descriptor `fd` in the file descriptor set `fdset`.

int FD_ISSET(int fd, fd_set * fdset)
        Returns a non-zero value if the bit for the file descriptor `fd` is set in the file descriptor set by `fdset`, and 0 otherwise.

void FD_SET(int fd, fd_set * fdset)
        Sets the bit for the file descriptor `fd` in the file descriptor set `fdset`.

void FD_ZERO(fd_set * fdset)
        Initialises the file descriptor set `fdset` to have zero bits for all file descriptors.
```

`FD_SETSIZE`
Maximum number of file descriptors in an `fd_set` structure.

The following are declared as functions and may also be defined as macros:

```c
int getitimer(int which, struct itimerval *value);
int setitimer(int which, const struct itimerval *value,
                struct itimerval *o_value);
int gettimeofday(struct timeval *tp, void *tzp);
int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *errorfds, struct timeval *timeout);
int utimes(const char *path, const struct timeval times[2]);
```
SEE ALSO

getitimer(), gettime(), select(), setitimer(), utimes().

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
    sys/timeb.h — additional definitions for date and time

SYNOPSIS
    #include <sys/timeb.h>

DESCRIPTION
    The <sys/timeb.h> header defines the timeb structure that includes at least the following members:

    time_t       time       the seconds portion of the current time
    unsigned short millitm  the milliseconds portion of the current time
    short        timezone   the local timezone in minutes west of Greenwich
    short        dstflag    TRUE if Daylight Savings Time is in effect

    The time_t type is defined as described in <sys/types.h>.

    The header <sys/timeb.h> declares the following as a function which may also be defined as a macro:

        int   ftime(struct timeb *tp);

SEE ALSO
    ftime(), <time.h>.

CHANGE HISTORY
    First released in Issue 4, Version 2.
NAME
    sys/times.h — file access and modification times structure

SYNOPSIS
    #include <sys/times.h>

DESCRIPTION
    The <sys/times.h> header defines the structure tms, which is returned by times() and includes at least the following members:

    clock_t tms_utime  user CPU time
    clock_t tms_stime  system CPU time
    clock_t tms_cutime user CPU time of terminated child processes
    clock_t tms_cstime system CPU time of terminated child processes

    The clock_t type is defined as described in <sys/types.h>.

    The following is declared as a function and may also be defined as a macro:

    clock_t times(struct tms *buffer);

SEE ALSO
    times(), <sys/types.h>.

CHANGE HISTORY
    First released in Issue 1.
    Derived from Issue 1 of the SVID.

Issue 4
    The following change is incorporated for alignment with the ISO POSIX-1 standard:
    • The function declarations in this header are expanded to full ISO C prototypes.

    Other changes are incorporated as follows:
    • Reference to the header <sys/types.h> is added for the definitions of clock_t.
    • This issue states that the times() function can also be defined as a macro.
<sys/types.h>

NAME
sys/types.h — data types

SYNOPSIS
#include <sys/types.h>

DESCRIPTION
The <sys/types.h> header includes definitions for at least the following types:

EX
- clock_t Used for system times in clock ticks or CLOCK_PER_SEC (see <time.h>).
- dev_t Used for device IDs.
- gid_t Used for group IDs.

UX
- id_t Used as a general identifier; can be used to contain at least a pid_t, uid_t or a gid_t.
- ino_t Used for file serial numbers.

EX
- key_t Used for interprocess communication.
- mode_t Used for some file attributes.
- nlink_t Used for link counts.
- off_t Used for file sizes.
- pid_t Used for process IDs and process group IDs.
- size_t Used for sizes of objects.
- ssize_t Used for a count of bytes or an error indication.
- time_t Used for time in seconds.
- uid_t Used for user IDs.

UX
- useconds_t Used for time in microseconds.

EX
All of the types except key_t are defined as arithmetic types of an appropriate length. Additionally, size_t is an unsigned integral type, and pid_t, ssize_t and off_t are signed integral types. The type ssize_t is capable of storing values at least in the range from −1 to [SSIZE_MAX] inclusive. The type useconds_t is an unsigned integral type capable of storing values at least in the range zero to 1000000.

SEE ALSO
bsearch(), chmod(), chown(), closedir(), creat(), fcntl(), fstat(), getegid(), geteuid(), getgid(), getgroups(), getpgrp(), getpid(), getppid(), getpriority(), getuid(), kill(), lseek(), mkdir(), mkfifo(), msgctl(), msgget(), msgsnd(), msgrcv(), open(), opendir(), readdir(), rewinddir(), semctl(), semget(), semop(), setegid(), setgid(), setsid(), setuid(), shmat(), shmctl(), shmdt(), shmget(), stat(), tcgetpgrp(), tcsetpgrp(), umask(), utime(), waitid().

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated for alignment with the ISO POSIX-1 standard:
- The data type ssize_t is added.
- The DESCRIPTION section is expanded to indicate the required arithmetic types.

Other changes are incorporated as follows:
- The clock_t type is marked as an extension.
- In the last paragraph of the DESCRIPTION section, only the reference to type key_t is now marked as an extension.
Issue 4, Version 2

The \texttt{id_t} and \texttt{useconds_t} types are defined for X/Open UNIX conformance. The capability of the \texttt{useconds_t} type is described.
NAME

sys/uio.h — definitions for vector I/O operations

SYNOPSIS

UX

#include <sys/uio.h>

DESCRIPTION

The <sys/uio.h> header defines the iovec structure that includes at least the following members:

- void *iov_base  base address of a memory region for input or output
- size_t iov_len   the size of the memory pointed to by iov_base

The following are declared as functions and may also be defined as macros:

- ssize_t readv(int fildes, const struct iovec *iov, int iovcnt);
- ssize_t writev(int fildes, const struct iovec *iov, int iovcnt);

SEE ALSO

read(), write().

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
sys/utsname.h — system name structure

SYNOPSIS
#include <sys/utsname.h>

DESCRIPTION
The <sys/utsname.h> header defines structure utsname, which includes at least the following members:

    char sysname[]    name of this implementation of the operating system
    char nodename[]   name of this node within an implementation-dependent communications network
    char release[]     current release level of this implementation
    char version[]     current version level of this release
    char machine[]     name of the hardware type on which the system is running

The character arrays are of unspecified size, but the data stored in them is terminated by a null byte.

The following is declared as a function and may also be defined as a macro:

int uname (struct utsname *name);

SEE ALSO
uname().

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following change is incorporated for alignment with the ISO C standard:
    • The function declarations in this header are expanded to full ISO C prototypes.

Other changes are incorporated as follows:
    • The word “character” is replaced with the word “byte” in the DESCRIPTION section.
    • The function in this header can now also be defined as a macro.
NAME
sys/wait.h — declarations for waiting

SYNOPSIS
#include <sys/wait.h>

DESCRIPTION
The <sys/wait.h> header defines the following symbolic constants for use with waitpid():

- WNOHANG: Do not hang if no status is available, return immediately.

and the following macros for analysis of process status values:

- WEXITSTATUS(): Return exit status.
- WIFCONTINUED(): True if child has been continued
- WIFEXITED(): True if child exited normally.
- WIFSIGNALED(): True if child exited due to uncaught signal.
- WIFSTOPPED(): True if child is currently stopped.
- WSTOPSIG(): Return signal number that caused process to stop.
- WTERMSIG(): Return signal number that caused process to terminate.

The following symbolic constants are defined as possible values for the options argument to waitid():

- WEXITED: Wait for processes that have exited.
- WSTOPPED: Status will be returned for any child that has stopped upon receipt of a signal.
- WCONTINUED: Status will be returned for any child that was stopped and has been continued.
- WNOHANG: Return immediately if there are no children to wait for.
- WNOWAIT: Keep the process whose status is returned in infop in a waitable state.

The type idtype_t is defined as an enumeration type whose possible values include at least the following:

- P_ALL
- P_PID
- P_PGID

The id_t type is defined as described in <sys/types.h>.

The siginfo_t type is defined as described in <signal.h>.

The rusage structure is defined as described in <sys/resource.h>.

The pid_t type is defined as described in <sys/types.h>.

Inclusion of the <sys/wait.h> header may also make visible all symbols from <signal.h> and <sys/resource.h>.

The following are declared as functions and may also be defined as macros:

- pid_t wait(int *stat_loc);
- pid_t wait3(int *stat_loc, int options, struct rusage *resource_usage);
- int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);
- pid_t waitpid(pid_t pid, int *stat_loc, int options);

SEE ALSO
wait(), wait3(), waitid(). <sys/resource.h>, <sys/types.h>.
CHANGE HISTORY

First released in Issue 3.

Entry included for alignment with the POSIX.1-1988 standard.

Issue 4

The following change is incorporated for alignment with the ISO POSIX-1 standard:

- The function declarations in this header are expanded to full ISO C prototypes.

Another change is incorporated as follows:

- Reference to the header `<sys/types.h>` is added for the definition of `pid_t` and marked as an extension.

Issue 4, Version 2

The following changes are incorporated for X/OPEN UNIX conformance:

- The WIFCONTINUED macro, the list of symbolic constants for the `options` argument to `waitid()`, and the description of the `idtype_t` enumeration type are added.

- A statement is added indicated that inclusion of this header may also make visible constants from `<signal.h>` and `<sys/resource.h>`.

- The `wait3()` and `waitid()` functions are added to the list of functions declared in this header.
NAME

`tar.h` — extended tar definitions

SYNOPSIS

```c
#include <tar.h>
```

DESCRIPTION

Header block definitions are:

General definitions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMAGIC</td>
<td>&quot;ustar&quot;</td>
<td>ustar plus null byte.</td>
</tr>
<tr>
<td>TMAIDLEN</td>
<td>6</td>
<td>Length of the above.</td>
</tr>
<tr>
<td>TVERSION</td>
<td>&quot;00&quot;</td>
<td>00 without a null byte.</td>
</tr>
<tr>
<td>TVERSLEN</td>
<td>2</td>
<td>Length of the above.</td>
</tr>
</tbody>
</table>

Typeflag field definitions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGTYPE</td>
<td>'0'</td>
<td>Regular file.</td>
</tr>
<tr>
<td>AREGTYPE</td>
<td>'0'</td>
<td>Regular file.</td>
</tr>
<tr>
<td>LNKTYPE</td>
<td>'1'</td>
<td>Link.</td>
</tr>
<tr>
<td>SYMTYPE</td>
<td>'2'</td>
<td>Symbolic link.</td>
</tr>
<tr>
<td>CHRTYPE</td>
<td>'3'</td>
<td>Character special.</td>
</tr>
<tr>
<td>BLKTYPE</td>
<td>'4'</td>
<td>Block special.</td>
</tr>
<tr>
<td>DIRTYPE</td>
<td>'5'</td>
<td>Directory.</td>
</tr>
<tr>
<td>FIFOTYPE</td>
<td>'6'</td>
<td>FIFO special.</td>
</tr>
<tr>
<td>CONTTYPE</td>
<td>'7'</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

Mode field bit definitions (octal):

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSUID</td>
<td>04000</td>
<td>Set UID on execution.</td>
</tr>
<tr>
<td>TSGID</td>
<td>02000</td>
<td>Set GID on execution.</td>
</tr>
<tr>
<td>TSVTX</td>
<td>01000</td>
<td>On directories, restricted deletion flag.</td>
</tr>
<tr>
<td>TUREAD</td>
<td>00400</td>
<td>Read by owner.</td>
</tr>
<tr>
<td>TUWRITE</td>
<td>00200</td>
<td>Write by owner special.</td>
</tr>
<tr>
<td>TUEXEC</td>
<td>00100</td>
<td>Execute/search by owner.</td>
</tr>
<tr>
<td>TGREAD</td>
<td>00040</td>
<td>Read by group.</td>
</tr>
<tr>
<td>TGWRITE</td>
<td>00020</td>
<td>Write by group.</td>
</tr>
<tr>
<td>TGEXEC</td>
<td>00010</td>
<td>Execute/search by group.</td>
</tr>
<tr>
<td>TOREAD</td>
<td>00004</td>
<td>Read by other.</td>
</tr>
<tr>
<td>TOWRITE</td>
<td>00002</td>
<td>Write by other.</td>
</tr>
<tr>
<td>TOEXEC</td>
<td>00001</td>
<td>Execute/search by other.</td>
</tr>
</tbody>
</table>

SEE ALSO

The XCU specification, `tar`.

CHANGE HISTORY

First released in Issue 3.

Derived from the entry in the POSIX.1-1988 standard.

Issue 4

This entry is moved from the referenced `Headers` specification.
Issue 4, Version 2

The following changes are incorporated for X/OPEN UNIX conformance:

- The significance of SYMTYPE as the value of the typeflag field is explained.
- The value of TSVTX as the value of the mode field is explained.
NAME
termios.h — define values for termios

SYNOPSIS
#include <termios.h>

DESCRIPTION
The <termios.h> header contains the definitions used by the terminal I/O interfaces (see the XBD specification, Chapter 9, General Terminal Interface for the structures and names defined).

The termios Structure
The following data types are defined through typedef:

cc_t Used for terminal special characters.
speed_t Used for terminal baud rates.
tcflag_t Used for terminal modes.

The above types are all unsigned integral types.

The termios structure is defined, and includes at least the following members:

tcflag_t c_iflag input modes
tcflag_t c_oflag output modes
tcflag_t c_cflag control modes
tcflag_t c_lflag local modes
cc_t c_cc[NCCS] control chars

A definition is given for:

NCCS Size of the array c_cc for control characters.

The following subscript names for the array c_cc are defined:

<table>
<thead>
<tr>
<th>Subscript Usage</th>
<th>Canonical Mode</th>
<th>Non-Canonical Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VEOF</td>
<td></td>
<td>EOF character</td>
<td></td>
</tr>
<tr>
<td>VEO</td>
<td></td>
<td>EOL character</td>
<td></td>
</tr>
<tr>
<td>VERASE</td>
<td></td>
<td>ERASE character</td>
<td></td>
</tr>
<tr>
<td>VINTR</td>
<td>VINTR</td>
<td>INTR character</td>
<td></td>
</tr>
<tr>
<td>VKILL</td>
<td>VMIN</td>
<td>KILL character</td>
<td></td>
</tr>
<tr>
<td>VQUIT</td>
<td>VQUIT</td>
<td>QUIT character</td>
<td></td>
</tr>
<tr>
<td>VSTART</td>
<td>VSTART</td>
<td>START character</td>
<td></td>
</tr>
<tr>
<td>VSTOP</td>
<td>VSTOP</td>
<td>STOP character</td>
<td></td>
</tr>
<tr>
<td>VSUSP</td>
<td>VSUSP</td>
<td>SUSP character</td>
<td></td>
</tr>
<tr>
<td>VTIME</td>
<td>VTIME</td>
<td>TIME character</td>
<td></td>
</tr>
</tbody>
</table>

The subscript values are unique, except that the VMIN and VTIME subscripts may have the same values as the VEOF and VEOL subscripts, respectively.
## Input Modes

The `c_iflag` field describes the basic terminal input control:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRKINT</td>
<td>Signal interrupt on break.</td>
</tr>
<tr>
<td>ICRNL</td>
<td>Map CR to NL on input.</td>
</tr>
<tr>
<td>IGNBRK</td>
<td>Ignore break condition.</td>
</tr>
<tr>
<td>IGNCR</td>
<td>Ignore CR</td>
</tr>
<tr>
<td>IGNPAR</td>
<td>Ignore characters with parity errors.</td>
</tr>
<tr>
<td>INLCR</td>
<td>Map NL to CR on input.</td>
</tr>
<tr>
<td>INPCK</td>
<td>Enable input parity check.</td>
</tr>
<tr>
<td>ISTRIP</td>
<td>Strip character</td>
</tr>
<tr>
<td>IUCLC</td>
<td>Map upper-case to lower-case on input (TO BE WITHDRAWN).</td>
</tr>
<tr>
<td>IXANY</td>
<td>Enable any character to restart output.</td>
</tr>
<tr>
<td>IXOFF</td>
<td>Enable start/stop input control.</td>
</tr>
<tr>
<td>IXON</td>
<td>Enable start/stop output control.</td>
</tr>
<tr>
<td>PARMRK</td>
<td>Mark parity errors.</td>
</tr>
</tbody>
</table>

## Output Modes

The `c_oflag` field specifies the system treatment of output:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPOST</td>
<td>Post-process output</td>
</tr>
<tr>
<td>OLCUC</td>
<td>Map lower-case to upper-case on output (TO BE WITHDRAWN).</td>
</tr>
<tr>
<td>ONLCR</td>
<td>Map NL to CR-NL on output.</td>
</tr>
<tr>
<td>OCRNL</td>
<td>Map CR to NL on output.</td>
</tr>
<tr>
<td>ONOCR</td>
<td>No CR output at column 0.</td>
</tr>
<tr>
<td>ONLRET</td>
<td>NL performs CR function.</td>
</tr>
<tr>
<td>OFILL</td>
<td>Use fill characters for delay.</td>
</tr>
<tr>
<td>NLDLY</td>
<td>Select newline delays:</td>
</tr>
<tr>
<td></td>
<td>NL0</td>
</tr>
<tr>
<td></td>
<td>NL1</td>
</tr>
<tr>
<td>CRDLY</td>
<td>Select carriage-return delays:</td>
</tr>
<tr>
<td></td>
<td>CR0</td>
</tr>
<tr>
<td></td>
<td>CR1</td>
</tr>
<tr>
<td></td>
<td>CR2</td>
</tr>
<tr>
<td></td>
<td>CR3</td>
</tr>
<tr>
<td>TABDLY</td>
<td>Select horizontal-tab delays:</td>
</tr>
<tr>
<td></td>
<td>TAB0</td>
</tr>
<tr>
<td></td>
<td>TAB1</td>
</tr>
<tr>
<td></td>
<td>TAB2</td>
</tr>
<tr>
<td></td>
<td>TAB3</td>
</tr>
<tr>
<td>BSDLY</td>
<td>Select backspace delays:</td>
</tr>
<tr>
<td></td>
<td>BS0</td>
</tr>
<tr>
<td></td>
<td>BS1</td>
</tr>
<tr>
<td>VTDLY</td>
<td>Select vertical-tab delays:</td>
</tr>
<tr>
<td></td>
<td>VT0</td>
</tr>
<tr>
<td></td>
<td>VT1</td>
</tr>
</tbody>
</table>
**<termios.h>**

<table>
<thead>
<tr>
<th>FFDLY</th>
<th>Select form-feed delays:</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF0</td>
<td>Form-feed delay type 0.</td>
</tr>
<tr>
<td>FF1</td>
<td>Form-feed delay type 1.</td>
</tr>
</tbody>
</table>

**Baud Rate Selection**

The input and output baud rates are stored in the `termios` structure. These are the valid values for objects of type `speed_t`. The following values are defined, but not all baud rates need be supported by the underlying hardware.

- B0: Hang up
- B50: 50 baud
- B75: 75 baud
- B110: 110 baud
- B134: 134.5 baud
- B150: 150 baud
- B200: 200 baud
- B300: 300 baud
- B600: 600 baud
- B1200: 1200 baud
- B1800: 1800 baud
- B2400: 2400 baud
- B4800: 4800 baud
- B9600: 9600 baud
- B19200: 19200 baud
- B38400: 38400 baud

**Control Modes**

The `c_cflag` field describes the hardware control of the terminal; not all values specified are required to be supported by the underlying hardware:

- **CSIZE** (Character size):
  - CS5: 5 bits.
  - CS6: 6 bits.
  - CS7: 7 bits.
  - CS8: 8 bits.

- **CSTOPB**: Send two stop bits, else one.
- **CREAD**: Enable receiver.
- **PARENB**: Parity enable.
- **PARODD**: Odd parity, else even.
- **HUPCL**: Hang up on last close.
- **CLOCAL**: Ignore modem status lines.
Local Modes

The \texttt{c_lflag} field of the argument structure is used to control various terminal functions:

\begin{itemize}
  \item \texttt{ECHO} Enable echo.
  \item \texttt{ECHOE} Echo erase character as error-correcting backspace.
  \item \texttt{ECHOK} Echo KILL.
  \item \texttt{ECHONL} Echo NL.
  \item \texttt{ICANON} Canonical input (erase and kill processing).
  \item \texttt{IEXTEN} Enable extended input character processing.
  \item \texttt{ISIG} Enable signals.
  \item \texttt{NOFLSH} Disable flush after interrupt or quit.
  \item \texttt{TOSTOP} Send SIGTTOU for background output.
  \item \texttt{XCASE} Canonical upper/lower presentation (TO BE WITHDRAWN).
\end{itemize}

Attribute Selection

The following symbolic constants for use with \texttt{tcsetattr()} are defined:

\begin{itemize}
  \item \texttt{TCSANOW} Change attributes immediately.
  \item \texttt{TCSADRAIN} Change attributes when output has drained.
  \item \texttt{TCSAFLUSH} Change attributes when output has drained; also flush pending input.
\end{itemize}

Line Control

The following symbolic constants for use with \texttt{tcflush()} are defined:

\begin{itemize}
  \item \texttt{TCIFLUSH} Flush pending input. Flush untransmitted output.
  \item \texttt{TCIOFLUSH} Flush both pending input and untransmitted output.
\end{itemize}

The following symbolic constants for use with \texttt{tcflow()} are defined:

\begin{itemize}
  \item \texttt{TCIOFF} Transmit a STOP character, intended to suspend input data.
  \item \texttt{TCION} Transmit a START character, intended to restart input data.
  \item \texttt{TCOOFF} Suspend output.
  \item \texttt{TCOON} Restart output.
\end{itemize}

The following are declared as functions and may also be defined as macros:

\begin{itemize}
  \item \texttt{speed_t cfgetispeed(const struct termios *termios_p)};
  \item \texttt{speed_t cfgetospeed(const struct termios *termios_p)};
  \item \texttt{int cfsetispeed(struct termios *termios_p, speed_t speed)};
  \item \texttt{int cfsetospeed(struct termios *termios_p, speed_t speed)};
  \item \texttt{int tcdrain(int fildes)};
  \item \texttt{int tcflow(int fildes, int action)};
  \item \texttt{int tcflush(int fildes, int queue_selector)};
  \item \texttt{int tcgetattr(int fildes, struct termios *termios_p)};
  \item \texttt{pid_t tcgetsid(int fildes)};
  \item \texttt{int tcsendbreak(int fildes, int duration)};
  \item \texttt{int tcsetattr(int fildes, int optional_actions, struct termios *termios_p)};
\end{itemize}
APPLICATION USAGE

The following names are commonly used as extensions to the above. They are therefore reserved and portable applications should not use them.

<table>
<thead>
<tr>
<th>CBAUD</th>
<th>EXTB</th>
<th>VDSUSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFECO</td>
<td>FLUSHO</td>
<td>VLNEXT</td>
</tr>
<tr>
<td>ECHOCTL</td>
<td>LOBLK</td>
<td>VREPRINT</td>
</tr>
<tr>
<td>ECHOKE</td>
<td>PENDIN</td>
<td>VSTATUS</td>
</tr>
<tr>
<td>ECHOPRT</td>
<td>SWTCH</td>
<td>VWERASE</td>
</tr>
<tr>
<td>EXTA</td>
<td>VDISCARD</td>
<td></td>
</tr>
</tbody>
</table>

SEE ALSO

cfgetispeed(), cfgetospeed(), cfsetispeed(), cfsetospeed(), tcdrain(), tcflow(), tcflush(), tcgetattr(), tcgetsid(), tcsetbreak(), tcsetattr(), the XBD specification, Chapter 9, General Terminal Interface.

CHANGE HISTORY

First released in Issue 3.

Entry included for alignment with the ISO POSIX-1 standard.

Issue 4

The following changes are incorporated for alignment with the ISO POSIX-1 standard:

- The function declarations in this header are expanded to full ISO C prototypes.
- Some minor rewording of the DESCRIPTION section is done to align the text more exactly with the ISO POSIX-1 standard. No functional differences are implied by these changes.
- The list of mask name symbols for the c_oflag field have all been marked as extensions, with the exception of OPOST.

Other changes are incorporated as follows:

- The following words are removed from the description of the c_cc array:
  
  “Implementations that do not support the job control option, may ignore the SUSP character value in the c_cc array indexed by the VSUSP subscript.”

  This is because job control is defined as mandatory for Issue 4 conforming implementations.

- The mask name symbols IUCLC and OLCUC are marked TO BE WITHDRAWN.

Issue 4, Version 2

For X/OPEN UNIX conformance, the tcgetsid() function is added to the list of functions declared in this header.
NAME

time.h — time types

SYNOPSIS

#include <time.h>

DESCRIPTION

The <time.h> header declares the structure tm, which includes at least the following members:

- int tm_sec [0,61]
- int tm_min [0,59]
- int tm_hour [0,23]
- int tm_mday [1,31]
- int tm_mon [0,11]
- int tm_year [1900, year]
- int tm_wday [0,6] (Sunday = 0)
- int tm_yday [0,365]
- int tm_isdst [TO BE WITHDRAWN]

The value of tm_isdst is positive if Daylight Saving Time is in effect, 0 if Daylight Saving Time is not in effect, and negative if the information is not available.

This header defines the following symbolic names:

- NULL Null pointer constant.
- CLK_TCK Number of clock ticks per second returned by the times() function (TO BE WITHDRAWN).
- CLOCKS_PER_SEC A number used to convert the value returned by the clock() function into seconds.

The clock_t, size_t and time_t types are defined as described in <sys/types.h>.

Although the value of CLOCKS_PER_SEC is required to be 1 million on all XSI-conformant systems, it may be variable on other systems and it should not be assumed that CLOCKS_PER_SEC is a compile-time constant.

The value of CLK_TCK is currently the same as the value of sysconf(_SC_CLK_TCK); however, new applications should call sysconf() because the CLK_TCK macro will be withdrawn in a future version of this document.

The <time.h> header provides a declaration for getdate_err.

The following are declared as functions and may also be defined as macros:

- char *asctime(const struct tm *timeptr);
- clock_t clock(void);
- char *ctime(const time_t *clock);
- double difftime(time_t time1 , time_t time0);
- struct tm *getdate(const char *string);
- struct tm *gmtime(const time_t *timer);
- struct tm *localtime(const time_t *timer);
- time_t mktime(struct tm *timeptr);
- size_t strftime(char * s , size_t maxsize , const char * format ,
                  const struct tm * timptr);
- char *strptime(const char * buf , const char * format,
                 struct tm *tm);
time_t time(time_t *tloc);
void tzset(void);

The following are declared as variables:

extern int daylight;
extern long int timezone;
extern char *tzname[];

APPLICATION USAGE
The range [0,61] for \texttt{tm_sec} allows for the occasional leap second or double leap second.

SEE ALSO
asctime(), clock(), ctime(), daylight, difftime(), getdate(), gmtime(), localtime(), mktime(), strftime(), strptime(), sysconf(), time(), timezone, tzname(), tzset(), utime().

CHANGE HISTORY
First released in Issue 1.
Derived from Issue 1 of the SVID.

Issue 4
The following changes are incorporated for alignment with the ISO C standard:

\begin{itemize}
  \item The function declarations in this header are expanded to full ISO C prototypes.
  \item The range of \texttt{tm_min} is changed from [0,61] to [0,59].
  \item Possible settings of \texttt{tm_isdst} and their meanings are added.
  \item The functions \texttt{clock()} and \texttt{difftime()} are added to the list of functions declared in this header.
\end{itemize}

Other changes are incorporated as follows:

\begin{itemize}
  \item The symbolic name CLK_TCK is marked as an extension and TO BE WITHDRAWN. Warnings about its use are also added to the DESCRIPTION section.
  \item Reference to the header <sys/types.h> is added for the definitions of \texttt{clock_t}, \texttt{size_t} and \texttt{time_t}.
  \item References to CLK_TCK are changed to CLOCKS_PER_SEC in part of the DESCRIPTION section. The fact that CLOCKS_PER_SEC is always one millionth of a second on XSI-conformant systems is also marked as an extension.
  \item External declarations for \texttt{daylight}, \texttt{timezone} and \texttt{tzname} are added. The first two are marked as extensions.
  \item The function \texttt{strptime()} is added to the list of functions declared in this header.
  \item A note about the settings of \texttt{tm_sec} is added to the APPLICATION USAGE section.
\end{itemize}

Issue 4, Version 2
The following changes are incorporated for X/OPEN UNIX conformance:

\begin{itemize}
  \item The <time.h> header provides a declaration for \texttt{getdate_err}.
  \item The \texttt{getdate()} function is added to the list of functions declared in this header.
\end{itemize}
NAME

ucontext — user context

SYNOPSIS

UX

```
#include <ucontext.h>
```

DESCRIPTION

The `<ucontext.h>` header defines the `mcontext_t` type through `typedef`.

The `<ucontext.h>` header defines the `ucontext_t` type as a structure that includes at least the following members:

```c
ucontext_t *uc_link
sigset_t uc_sigmask
stack_t uc_stack
mcontext_t uc_mcontext
```

- `uc_link`: pointer to the context that will be resumed when this context returns
- `uc_sigmask`: the set of signals that are blocked when this context is active
- `uc_stack`: the stack used by this context
- `uc_mcontext`: a machine-specific representation of the saved context

The types `sigset_t` and `stack_t` are defined as in `<signal.h>`.

The following are declared as functions and may also be defined as macros:

```c
int getcontext(ucontext_t *ucp);
int setcontext(const ucontext_t *ucp);
void makecontext(ucontext_t *ucp, (void *func)(), int argc, ...);
int swapcontext(ucontext_t *oucp, const ucontext_t *ucp);
```

SEE ALSO

`getcontext()`, `makecontext()`, `sigaction()`, `sigprocmask()`, `sigaltstack()`, `<signal.h>`.

CHANGE HISTORY

NAME
ulimit.h — ulimit commands

SYNOPSIS
EX
```
#include <ulimit.h>
```

DESCRIPTION
The <ulimit.h> header defines the symbolic constants used in the ulimit() function.

Symbolic constants:
UL_GETFSIZE Get maximum file size.
UL_SETFSIZE Set maximum file size.

The following is declared as a function and may also be defined as a macro:
```
long int ulimit ( int cmd, ...);
```

SEE ALSO
ulimit().

CHANGE HISTORY
First released in Issue 3.

Issue 4
The following change is incorporated in this issue:
- The function declarations in this header are expanded to full ISO C prototypes.
NAME
unistd.h — standard symbolic constants and types

SYNOPSIS
#include <unistd.h>

DESCRIPTION
The <unistd.h> header defines miscellaneous symbolic constants and types, and declares miscellaneous functions. The contents of this header are shown below.

Version Test Macros
The following symbolic constants are defined:

_POSIX_VERSION Integer value indicating version of the ISO POSIX-1 standard (C language binding).

_POSIX2_VERSION Integer value indicating version of the ISO POSIX-2 standard (Commands).

_POSIX2_C_VERSION Integer value indicating version of the ISO POSIX-2 standard (C language binding) and whether the X/Open POSIX2 C-language Binding Feature Group is supported.

_EX
_XOPEN_VERSION Integer value indicating version of the X/Open Portability Guide to which the implementation conforms.

_EX

_POSIX_VERSION is defined in the ISO POSIX-1 standard. It changes with each new version of the ISO POSIX-1 standard.

_POSIX2_VERSION is defined in the ISO POSIX-2 standard. It changes with each new version of the ISO POSIX-2 standard.

_POSIX2_C_VERSION is defined in the ISO POSIX-2 standard. It changes with each new version of the ISO POSIX-2 standard. When the C language binding option of the ISO POSIX-2 standard and therefore the X/Open POSIX2 C-language Binding Feature Group is not supported, _POSIX2_C_VERSION will be set to −1.

_EX

_XOPEN_VERSION is defined as an integer value greater than or equal to 4, indicating one of the issues of the X/Open Portability Guide to which the implementation conforms.

_XOPEN_XCU_VERSION is defined as an integer value indicating the version of the XCU specification to which the implementation conforms. If the value is −1, no commands and utilities are provided on the implementation. If the value is greater than or equal to 4, the functionality associated with the following symbols is also supported (see Mandatory Symbolic Constants on page 856 and Constants for Options and Feature Groups on page 856):

_POSIX2_C_BIND
_POSIX2_C_VERSION
_POSIX2_CHAR_TERM
_POSIX2_LOCALEDEF
_POSIX2_UPE
_POSIX2_VERSION

If this constant is not defined use the sysconf() function to determine which features are supported.
Each of the following symbolic constants is defined only if the implementation supports the indicated revision of the X/Open Portability Guide:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
</table>

**Mandatory Symbolic Constants**

**FIPS** Although all implementations conforming to this document support all of the FIPS features described below, there may be system-dependent or file-system-dependent configuration procedures that can remove or modify any or all of these features. Such configurations should not be made if strict FIPS compliance is required.

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

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_POSIX_CHOWN_RESTRICTED</td>
<td>The use of <code>chown()</code> is restricted to a process with appropriate privileges, and to changing the group ID of a file only to the effective group ID of the process or to one of its supplementary group IDs.</td>
</tr>
<tr>
<td>_POSIX_NO_TRUNC</td>
<td>Pathname components longer than <code>NAME_MAX</code> generate an error.</td>
</tr>
<tr>
<td>_POSIX_VDISABLE</td>
<td>Terminal special characters defined in <code>&lt;termios.h&gt;</code> can be disabled using this character value.</td>
</tr>
<tr>
<td>_POSIX_SAVED_IDS</td>
<td>Each process has a saved set-user-ID and a saved set-group-ID.</td>
</tr>
<tr>
<td>_POSIX_JOB_CONTROL</td>
<td>Implementation supports job control.</td>
</tr>
</tbody>
</table>

_POSIX_CHOWN_RESTRICTED, _POSIX_NO_TRUNC and _POSIX_VDISABLE will have values other than −1.

**Constants for Options and Feature Groups**

The following symbolic constants are defined to have the value −1 if the implementation will never provide the feature, and to have a value other than −1 if the implementation always provides the feature. If these are undefined, the `sysconf()` function can be used to determine whether the feature is provided for a particular invocation of the application.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_POSIX2_C_BIND</td>
<td>Implementation supports the C language binding option.</td>
</tr>
<tr>
<td>_POSIX2_C_DEV</td>
<td>Implementation supports the C language development utilities option.</td>
</tr>
<tr>
<td>_POSIX2_CHAR_TERM</td>
<td>Implementation supports at least one terminal type.</td>
</tr>
<tr>
<td>_POSIX2_FORT_DEV</td>
<td>Implementation supports the FORTRAN Development Utilities Option.</td>
</tr>
</tbody>
</table>
Headers

 siè <unistd.h>

_POSIX2_FORT_RUN
Implementation supports the FORTRAN Run-time Utilities Option.

EX
_POSIX2_LOCALEDEF
Implementation supports the creation of locales by the localedef utility.

_POSIX2_SW_DEV
Implementation supports the Software Development Utilities Option.

_POSIX2_UPE
The implementation supports the User Portability Utilities Option.

_XOPEN_CRYPT
The implementation supports the X/Open Encryption Feature Group.

_XOPEN_ENH_I18N
The implementation supports the X/Open Enhanced Internationalisation Feature Group.

_XOPEN_SHM
The implementation supports the X/Open Shared Memory Feature Group.

Constants for Functions

The following symbolic constant is defined:

NULL Null pointer

The following symbolic constants are defined for the access() function:

R_OK Test for read permission.
W_OK Test for write permission.
X_OK Test for execute (search) permission.
F_OK Test for existence of file.

The constants F_OK, R_OK, W_OK and X_OK and the expressions R_OK | W_OK, R_OK | X_OK and R_OK | W_OK | X_OK all have distinct values.

The following symbolic constant is defined for the confstr() function:

_CS_PATH If the ISO POSIX-2 standard is supported, this is the value for the PATH environment variable that finds all standard utilities. Otherwise the meaning of this value is unspecified.

The following symbolic constants are defined for the lseek() and fcntl() functions (they have distinct values):

SEEK_SET Set file offset to offset.
SEEK_CUR Set file offset to current plus offset.
SEEK_END Set file offset to EOF plus offset.

The following symbolic constants are defined for sysconf():

_SC_2_C_BIND
_SC_2_C_DEV
_SC_2_C_VERSION
_SC_2_FORT_DEV
_SC_2_FORT_RUN
_SC_2_LOCALEDEF
_SC_2_SW_DEV
_SC_2_UPE
_SC_2_VERSION
_SC_ARG_MAX

UX
_SC_ATEXIT_MAX
_SC_BC_BASE_MAX
_SC_BC_DIM_MAX
_SC_BC_SCALE_MAX

System Interfaces and Headers Issue 4, Version 2
The two constants \_SC_PAGESIZE and \_SC_PAGE_SIZE may be defined to have the same value.

The following symbolic constants are defined as possible values for the function argument to the lockf() function:

- F_LOCK: Lock a section for exclusive use.
- F_ULOCK: Unlock locked sections.
- F_TEST: Test section for locks by other processes.
- F_TLOCK: Test and lock a section for exclusive use.

The following symbolic constants are defined for pathconf():

- \_PC_CHOWN_RESTRICTED
- \_PC_LINK_MAX
- \_PC_MAX_CANON
- \_PC_MAX_INPUT
- \_PC_NAME_MAX
- \_PC_NO_TRUNC
- \_PC_PATH_MAX
- \_PC_PIPE_BUF
- \_PC_VDISABLE
The following symbolic constants are defined for file streams:

- STDIN_FILENO: File number of stdin. It is 0.
- STDOUT_FILENO: File number of stdout. It is 1.
- STDERR_FILENO: File number of stderr. It is 2.

**Type Definitions**

- The `size_t`, `ssize_t`, `uid_t`, `gid_t`, `off_t` and `pid_t` types are defined as described in `<sys/types.h>`.
- The `useconds_t` type is defined as described in `<sys/types.h>`.

**Declarations**

The following are declared as functions and may also be defined as macros:

```c
int access(const char *path, int amode);
unsigned int alarm(unsigned int seconds);
int brk(void *addr);
int chdir(const char *path);
int chown(const char *path, uid_t owner, gid_t group);
int chroot(const char *path);  // TO BE WITHDRAWN
int close(int fildes);
size_t confstr(int name, char *buf, size_t len);
int crypt(const char *key, const char *salt);
char *ctermid(char *s);
int fchown(int fildes, uid_t owner, gid_t group);
int fchdir(int fildes);
void fchdir(int fildes);
long int fpathconf(int fildes, int name);
int fsync(int fildes);
int ftruncate(int fildes, off_t length);
char *getcwd(char *buf, size_t size);
int getdtablesize(void);
gid_t getegid(void);
uid_t geteuid(void);
gid_t getgid(void);
int getgroups(int gidsetsize, gid_t grouplist[]);
long int gethostid(void);
char *getlogin(void);
int getopt(int argc, char *const argv[], const char *optstring);
int getpagesize(void);
char *getpass(const char *prompt);  // TO BE WITHDRAWN
```
The following external variables are declared:

extern char *optarg;
extern int optind, opterr, optopt;

SEE ALSO

access(), alarm(), brk(), chdir(), chown(), chroot(), close(), crypt(), ctermid(), cuserid(), dup(), encrypt(), environ(), exec(), exit(), fchdir(), fchown(), fcntl(), fork(), fpathconf(), fsync(), ftruncate(), getcwd(), getdtablesize(), getegid(), geteuid(), getgid(), getgroups(), gethostid(), getlogin(), getpagesize(), getpass(), getpgid(), getpgrp(), getpid(), getppid(), getsid(), getuid(), getwdir(), isatty(), lchown(), link(), lockf(), lseek(), nice(), pathconf(), pause(), pipe(), read(), readlink(), rmdir(), sbrk(), setegid(), setgid(), setuid(), setsid(), setsockopt(), sleep(), swab(), symlink(), symlink(), sysconf(), tcgetpgrp(), tcsetpgrp(), truncate(), ttyname(), ualarm(), unlink(), usleep(), vfork(), write(), <limits.h>, <sys/types.h>, <termios.h>, Section 1.2 on page 1.
CHANGE HISTORY

First released in Issue 1.

Derived from Issue 1 of the SVID.

Issue 4

The following changes are incorporated for alignment with the ISO POSIX-1 standard and the ISO POSIX-2 standard:

- The function declarations in this header are expanded to full ISO C prototypes.
- A large number of new constants are defined for the `sysconf()` function, including all those with prefixes _SC_2 and _SC_BC, plus:
  - `_SC_COLL_WEIGHTS_MAX`
  - `_SC_EXPR_NEST_MAX`
  - `_SC_LINE_MAX`
  - `_SC_RE_DUP_MAX`
  - `_SC_STREAM_MAX`
  - `_SC_TZNAME_MAX`
- The `confstr()` function is added to the list of functions declared in this header, complete with a new set of constants for alignment with the ISO POSIX-2 standard.

The following change is incorporated for alignment with the FIPS requirements:

- The following symbolic constants are always defined:
  - `_POSIX_CHOWN_RESTRICTED`
  - `_POSIX_NO_TRUNC`
  - `_POSIX_VDISABLE`
  - `_POSIX_SAVED_IDS`
  - `_POSIX_JOB_CONTROL`

In Issue 3, they are only defined if the associated option is present.

Other changes are incorporated as follows:

- The symbolic constants F_ULOCK, F_LOCK, F_TLOCK, F_TEST, GF_PATH, IF_PATH and PF_PATH are withdrawn.
- The required value of _XOPEN_VERSION is defined and the constant is marked as an extension.
- The constants _XOPEN_XPG2, _XOPEN_XPG3 and _XOPEN_XPG4 are added.
- The constants _POSIX2_* are added.
- Reference to the header `<sys/types.h>` is added for the definitions of `size_t`, `ssize_t`, `uid_t`, `gid_t`, `off_t` and `pid_t`. These are marked as extensions.
- The names `chroot()`, `crypt()`, `encrypt()`, `fsync()`, `getopt()`, `getpass()`, `nice()` and `swab()` are added to the list of functions declared in this header. With the exception of `getopt()`, these are all marked as extensions.
- The APPLICATION USAGE section is removed.

Issue 4, Version 2

The following changes are incorporated for X/OPEN UNIX conformance:

- The feature group constant _XOPEN_UNIX is defined.
• The `sysconf()` symbolic constants `_SC_ATEXIT_MAX`, `_SC_IOV_MAX`, `_SC_PAGESIZE` and `_SC_PAGE_SIZE` are defined.

• The `brk()`, `fchown()`, `fchdir()`, `ftruncate()`, `gethostid()`, `getpagesize()`, `getpgrp()`, `getpgid()`, `getresgid()`, `getuid()`, `gettimeofday()`, `lchown()`, `lockf()`, `readlink()`, `sbrk()`, `setgids()`, `setpgid()`, `setpgrp()`, `setreuid()`, `symlink()`, `sync()`, `truncate()`, `ualarm()`, `usleep()` and `vfork()` functions are added to the list of functions declared in this header.

• The symbolic constants `F_ULOCK`, `F_LOCK`, `F_TLOCK` and `F_TEST` are added.
NAME
utime.h — access and modification times structure

SYNOPSIS
#include <utime.h>

DESCRIPTION
The <utime.h> header declares the structure utimbuf, which includes the following members:

time_t actime access time
time_t modtime modification time

The times are measured in seconds since the Epoch.

EX
The type time_t is defined as described in <sys/types.h>.

The following is declared as a function and may also be defined as a macro:

int utime(const char *path, const struct utimbuf *times);

SEE ALSO
utime(), <sys/types.h>.

CHANGE HISTORY
First released in Issue 3.

Issue 4
The following change is incorporated for alignment with the ISO POSIX-1 standard:
• The function declarations in this header are expanded to full ISO C prototypes.

Another change is incorporated as follows:
• Reference to the header <sys/types.h> is added for the definition of time_t. This is marked as an extension.
NAME

utmpx.h — user accounting database definitions

SYNOPSIS

#include <utmpx.h>

DESCRIPTION

The <utmpx.h> header defines the utmpx structure that includes at least the following members:

- `char ut_user[]` user login name
- `char ut_id[]` unspecified initialisation process identifier
- `char ut_line[]` device name
- `pid_t ut_pid` process id
- `short int ut_type` type of entry
- `struct timeval ut_tv` time entry was made

The following symbolic constants are defined as possible values for the `ut_type` member of the utmpx structure:

- `EMPTY` No valid user accounting information.
- `BOOT_TIME` Identifies time of system boot.
- `OLD_TIME` Identifies time when system clock changed.
- `NEW_TIME` Identifies time after system clock changed.
- `USER_PROCESS` Identifies a process.
- `INIT_PROCESS` Identifies a process spawned by the init process.
- `LOGIN_PROCESS` Identifies the session leader of a logged in user.
- `DEAD_PROCESS` Identifies a session leader who has exited.

The following are declared as functions and may also be defined as macros:

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

SEE ALSO

`endutxent()`.

CHANGE HISTORY

First released in Issue 4, Version 2.
NAME
varargs.h — handle variable argument list (TO BE WITHDRAWN)

SYNOPSIS
EX
```
#include <varargs.h>
```

va_alist
va_dcl

void va_start(pvar)
va_list pvar;

type va_arg(pvar, type)
va_list pvar;

void va_end(pvar)
va_list pvar;

DESCRIPTION
The <varargs.h> header contains a set of macros which allows portable procedures that accept
variable argument lists to be written. Routines that have variable argument lists (such as
printf()) but do not use <varargs.h> are inherently non-portable, as different machines use
different argument-passing conventions.

va_alist Used as the parameter list in a function header.
va_dcl A declaration for va_alist. No semicolon should follow va_dcl.
va_list A type defined for the variable used to traverse the list.
va_start() Called to initialise pvar to the beginning of the list.
va_arg() Will return the next argument in the list pointed to by pvar. The argument
type is the type the argument is expected to be. Different types can be mixed,
but it is up to the routine to know what type of argument is expected, as it
cannot be determined at run time.

va_end() Used to clean up.

Multiple traversals, each bracketed by va_start() ... va_end(), are possible.
EXAMPLE
This example is a possible implementation of `execl()`.

```c
#include <varargs.h>

#define MAXARGS 100

/* execl is called by
   * execl(file, arg1, arg2, ... , (char *)0);
   */
execl(va_alist)
va_dcl
{
    va_list ap;
    char *file;
    char *args[MAXARGS];
    int argno = 0;

    va_start(ap);
    file = va_arg(ap, char *);
    while ((args[argno++] = va_arg(ap, char *)) != (char *)0)
        ;
    va_end(ap);
    return execv(file, args);
}
```

APPLICATION USAGE
It is up to the calling routine to specify how many arguments there are, since it is not always possible to determine this from the stack frame. For example, `execl()` is passed a zero pointer to signal the end of the list. The `printf()` function can tell how many arguments are there by the format.

It is non-portable to specify a second argument of `char`, `short` or `float` to `va_arg()`, since arguments seen by the called function are not type `char`, `short` or `float`. C language converts type `char` and `short` arguments to `int` and converts type `float` arguments to `double` before passing them to a function.

For backward compatibility with Issue 3, XSI-conformant systems support `<varargs.h>` as well as `<stdarg.h>`. Use of `<varargs.h>` is not recommended as this functionality is subject to future withdrawal.

SEE ALSO
`exec`, `printf()`.

CHANGE HISTORY
First released in Issue 1.

Issue 4
The following changes are incorporated in this issue:

- The interface is marked TO BE WITHDRAWN.

- The APPLICATION USAGE section is added, recommending use of `<stdarg.h>` in preference to this header.

- The FUTURE DIRECTIONS section is removed.
NAME
wchar.h — wide character types

SYNOPSIS
WP
#include <wchar.h>

DESCRIPTION
The <wchar.h> header defines the following data types through typedef:

wchar_t See <stddef.h>.
wint_t An integral type capable of storing any valid value of wchar_t, or WEOF.
wctype_t A scalar type of a data object that can hold values which represent locale-specific character classification.

EX
FILE As described in <stdio.h>.
size_t As described in <stddef.h>.

The <wchar.h> header declares the following as functions and may also define them as macros:

int iswalnum(wint_t wc);
int iswalpha(wint_t wc);
int iswcntrl(wint_t wc);
int iswdigit(wint_t wc);
int iswgraph(wint_t wc);
int iswlower(wint_t wc);
int iswprint(wint_t wc);
int iswpunct(wint_t wc);
int iswspace(wint_t wc);
int iswupper(wint_t wc);
int iswxdigit(wint_t wc);
int iswctype(wint_t wc, wctype_t prop);
wint_t fgetwc(FILE *stream);
wint_t *fgetws(wchar_t *s, int n, FILE *stream);
int fputwc(wint_t c, FILE *stream);
wint_t getwc(FILE *stream);
wint_t getwchar(void);
wchar_t *getws(wchar_t *s);
wint_t putwc(wint_t c, FILE *stream);
wint_t putwchar(wint_t c);
int putws(const wchar_t *s, FILE *stream);
wint_t towlower(wint_t wc);
wint_t towupper(wint_t wc);
wint_t ungetwc(wint_t c, FILE *stream);
wctype_t wctype(const char *property);
wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcschr(const wchar_t *ws, wchar_t wc);
int wcscmp(const wchar_t *ws1, const wchar_t *ws2);
int wcscoll(const wchar_t *ws1, const wchar_t *ws2);
wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2);
size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2);
szize_t wcsftime(wchar_t *ws, size_t maxsize,
const char *fmt, const struct tm *timptr);
szsize_t wcslen(const wchar_t *ws);
<wchar.h>

#define WEOF Constant expression of type wint_t that is returned by several WP functions to indicate end-of-file.

#define NULL As described in <stddef.h>.

Inclusion of the <wchar.h> header may make visible all symbols from the headers <ctype.h>, <stdio.h>, <stdlib.h>, <string.h>, <stdbool.h> and <time.h>.

SEE ALSO
iswalnum(), iswalpha(), iswcntrl(), iswdigit(), iswgraph(), iswlower(), iswprint(), iswpunct(), iswspace(), iswupper(), iswxdigit(), mblen(), mbstowcs(), mbtowc(), setlocale(), towlower(), towupper(), wcstombs(), wctomb(), <locale.h>.

CHANGE HISTORY
First released in Issue 4.
Derived from the MSE working draft.
NAME
wordexp.h — word-expansion types

SYNOPSIS
#include <wordexp.h>

DESCRIPTION
The <wordexp.h> header defines the structures and symbolic constants used by the wordexp() and wordfree() functions.

The structure type wordexp_t contains at least the following members:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_t</td>
<td>we_wordc — count of words matched by words</td>
</tr>
<tr>
<td>char **</td>
<td>we_wordv — pointer to list of expanded words</td>
</tr>
<tr>
<td>size_t</td>
<td>we_offs — slots to reserve at the beginning of we_wordv</td>
</tr>
</tbody>
</table>

The flags argument to the wordexp() function is the bitwise inclusive OR of the following flags:

- WRDE_APPEND: Append words to those previously generated.
- WRDE_DOOFHS: Number of null pointers to prepend to we_wordv.
- WRDE_NOCMD: Fail if command substitution is requested.
- WRDE_REUSE: The pwordexp argument was passed to a previous successful call to wordexp(), and has not been passed to wordfree(). The result will be the same as if the application had called wordfree() and then called wordexp() without WRDE_REUSE.
- WRDE_SHOWERR: Do not redirect stderr to /dev/null.
- WRDE_UNDEF: Report error on an attempt to expand an undefined shell variable.

The following constants are defined as error return values:

- WRDE_BADCHAR: One of the unquoted characters: `<newline> | & ; < > (){} appears in words in an inappropriate context.
- WRDE_BADVAL: Reference to undefined shell variable when WRDE_UNDEF is set in flags.
- WRDE_CMDSUB: Command substitution requested when WRDE_NOCMD was set in flags.
- WRDE_NOSPACE: Attempt to allocate memory failed.
- WRDE_NOSYS: The implementation does not support the function.
- WRDE_SYNTAX: Shell syntax error, such as unbalanced parentheses or unterminated string.

The following are declared as functions and may also be declared as macros:

```c
int wordexp(const char *words, wordexp_t *pwordexp, int flags);
void wordfree(wordexp_t *pwordexp);
```

The implementation may define additional macros or constants using names beginning with WRDE_.

SEE ALSO
wordexp(), the XCU specification.
CHANGE HISTORY
   First released in Issue 4.
   Derived from the ISO POSIX-2 standard.
Index

<assert.h> .................................................................746
<pio.h> ....................................................................747
<ctype.h> ...............................................................748
<dirent.h> ..............................................................749
<errno.h> ...............................................................750
<fcntl.h> ...............................................................753
<float.h> ..................................................................755
<mtmsg.h> .............................................................757
<nmatch.h> ............................................................759
<tfw.h> ....................................................................760
<glob.h> ....................................................................762
<grp.h> ....................................................................763
<conv.h> ....................................................................764
<langinfo.h> ..........................................................765
<libgen.h> ............................................................768
<limits.h> ...............................................................769
<locale.h> ...............................................................778
<math.h> ....................................................................780
<monetary.h> .........................................................783
<nl_types.h> ..........................................................785
<poll.h> .....................................................................786
<pwd.h> .....................................................................787
<regen.h> ..............................................................788
<regexp.h> ............................................................791
<re_comp.h> ............................................................790
<search.h> .............................................................792
<setjmp.h> .............................................................793
<signal.h> ..............................................................794
<stdarg.h> ............................................................801
<stddef.h> ............................................................803
<stdio.h> ..............................................................804
<stdlib.h> ............................................................807
<string.h> ............................................................810
<string.h> ............................................................812
<stropts.h> ...........................................................813
<sys/ipc.h> ...........................................................820
<sys/mman.h> .........................................................822
<sys/msg.h> ..........................................................823
<sys/resource.h> ....................................................825
<sys/sem.h> ...........................................................827
<sys/shm.h> ...........................................................829
<sys/stat.h> ...........................................................830
<sys/statvfs.h> .......................................................833
<sys/time.h> ..........................................................834
<sys/timeb.h> ..........................................................836
<sys/times.h> ..........................................................837
<sys/types.h> ..........................................................838
<sys/uio.h> ............................................................840
<sys/utsname.h> .......................................................841
<sys/wait.h> ...........................................................842
<syslog.h> .............................................................818
<tar.h> ....................................................................844
<termios.h> ............................................................846
<time.h> ....................................................................851
<timezone.h> ..........................................................853
<unistd.h> ..............................................................854
<unistd.h> ..............................................................855
<utime.h> ................................................................863
<utmpx.h> ...............................................................864
<vargraphs.h> .........................................................865
<wchar.h> ................................................................867
<wcwidth.h> ...........................................................869
<wordexp.h> ...........................................................869

±0

in floor() .................................................................170
in fmod() .................................................................172
_CS_PATH .................................................................857
_exit() .................................................................136
_IOFBF .....................................................................804
_IOLBF......................................................................804
_JONBF .....................................................................804
_longjmp() ..............................................................369
_PC constants
defined in <unistd.h> ...........................................858
used in pathconf() ..................................................429
_POSIX minimum values
in <limits.h> ..........................................................773
_POSIX2 constants in sysconf( ) .........................634
_POSIX2_CHAR_TERM ..............................................856
_POSIX2_C_BIND ....................................................856
_POSIX2_C_DEV ......................................................856
_POSIX2_C_VERSION ..................................................855
_POSIX2_FORT_DEV ..................................................856
_POSIX2_FORT_RUN ..................................................857
_POSIX2_LOCALDEFINE ...........................................857
_POSIX2_SW_DEV ......................................................857
_POSIX2_UPE ...........................................................857
_POSIX2_VERSION ......................................................855
in sysconf( ) ............................................................635
_POSIX_CHOWN_RESTRICTED 86, 429, 856, 861
_POSIX_JOB_CONTROL ...........................................634, 856, 861
_POSIX_NAME_MAX ...................................................774
Index

clock_t ................................................................. 838
close() .................................................................. 92
closedir() .......................................................... 94
closelog() ............................................................ 95
CODESET .......................................................... 765
COLL_WEIGHTS_MAX .......................... 634, 772
Common Usage C .............................................. 16
compilation environment ................................. 17
compile() ............................................................ 98
conformance .......................................................... 1
confstr() .............................................................. 99
control modes ...................................................... 848
control-normal .................................................. 460
CONTTYPE .......................................................... 844
conversion descriptor
  in exec .......................................................... 132, 134
  in iconv() .................................................. 294-295
  in iconv_close() ........................................ 296
  in iconv_open() ........................................... 297
conversion specification
  in fprintf() .................................................. 184-185, 187
  in fscanf() .................................................. 201-205
  in strftime() ............................................... 601-602
  in strftime() ............................................... 605-607
  in strftime() ............................................... 614-615
cos() .................................................................. 101
cosh() .................................................................. 102
CRDLY ................................................................. 847
CREAD ................................................................. 848
creat() ................................................................. 103
CRn ................................................................. 847
CRNCYSTR .......................................................... 766
crypt() ................................................................. 104
CSIZE ................................................................. 848
CSn ................................................................. 848
CSTOPB ............................................................... 848
ctermid() ........................................................... 105
cftime() ............................................................... 106
cuserid() ............................................................. 107
C_ constants in <cpio.h> ................................. 747
data structure
  dirent .............................................................. 749
  entry .............................................................. 792
group ................................................................. 763
iconv ................................................................. 778
msqid_ds ............................................................ 823
stat ................................................................. 830
tms ................................................................. 837
utimbuf ............................................................. 863
data type ............................................................. 38
ACTION ............................................................. 792
cc_t ................................................................. 846
DIR ................................................................. 749
div_t ................................................................. 807
ENTRY ............................................................... 792
FILE ................................................................. 804
fpos_t ................................................................. 804
glob_t ................................................................. 762
ldiv_t ................................................................. 807
msglen_t ............................................................ 823
msgqnum_t .......................................................... 823
nl_catd ............................................................... 785
nl_item ............................................................... 785
pthread_t ............................................................ 803
regex_t ............................................................... 788
regmatch_t ........................................................ 788
regoff_t ............................................................. 788
shmatt_t ............................................................. 829
sigset_t ............................................................. 794
sig_atomic_t ........................................................ 794
size_t ................................................................. 803
speed_t .............................................................. 846
tcflag_t .............................................................. 846
va_list ............................................................... 865
VISIT ................................................................. 792
wchar_t ............................................................... 803
wctype_t ............................................................. 867
wint_t ................................................................. 867
data types
  defined in <sys/types.h> ............................... 838
daylight ............................................................. 109
DAY_n ............................................................... 765
DBL_ constants
  defined in <float.h> ...................................... 756
dbm_clearerr() .................................................. 110
descriptor table
  returning size of ........................................... 234
dev_t ................................................................. 838
diffftime() .......................................................... 113
directive
  in fprintf() .................................................. 184
  in fscanf() .................................................. 201-202
  in strftime() ............................................... 614-615
directory
  returning pathname for current .................... 281
dirname() .......................................................... 114
DIRTYPE .......................................................... 844
div() ................................................................. 115
drand48() .......................................................... 116
dup() ................................................................. 118
D_FMT ............................................................... 765
D_T_FMT ........................................................... 765
Index

E2BIG ............................................................. 25, 750
in exec ............................................................. 133
in strfmon() ...................................................... 603
EACCES .......................................................... 25, 750
in access( ) ....................................................... 43
in catopen( ) ..................................................... 74
in chmod( ) ....................................................... 82
in chdir( ) ......................................................... 84
in chown( ) ....................................................... 86
in chroot( ) ....................................................... 88
in exec ............................................................. 133
in fopen( ) ....................................................... 178
in freopen( ) ................................................... 198
in ftw( ) .......................................................... 221
in getcwd( ) ..................................................... 229
in link( ) .......................................................... 355
in mkdir( ) ...................................................... 388
in mkfifo( ) ..................................................... 390
in open( ) ........................................................ 423
in opendir( ) ................................................... 425
in pathconf( ) ............................................... 430
in rename( ) .................................................... 490
in rmdir( ) ...................................................... 496
in setpgid( ) .................................................... 526
in shmat( ) ...................................................... 538
in shmctl( ) ..................................................... 540
in shmat( ) ...................................................... 543
in stat( ) ........................................................ 587
in unlink( ) ..................................................... 689
in utime( ) ...................................................... 693
in utimes( ) ..................................................... 695
EADDRINUSE ................................................ 25, 750
EADDRNOTAVAIL ........................................... 25, 750
EAFNOSUPPORT ............................................. 25, 750
EAGAIN ......................................................... 25, 750
in fclose( ) .................................................... 146
in fflush( ) .................................................... 159
in fgetc( ) ....................................................... 162
in fgetwc( ) .................................................... 166
in fork( ) ........................................................ 181
in fputc( ) ....................................................... 190
in fputwc( ) .................................................... 193
in fseek( ) ....................................................... 207
in fstat( ) ....................................................... 210
in fsync( ) ..................................................... 214
in ftruncate( ) ................................................ 215
in lseek( ) ....................................................... 375
in pathconf( ) ............................................... 430
in read( ) ........................................................ 461
in readdir( ) ................................................... 463
in readlink( ) .................................................. 537
in setbuf( ) ...................................................... 642
in tcdrain( ) .................................................... 644
in tcflow( ) ..................................................... 646
in tcflush( ) .................................................... 649
in tcgetattr( ) ............................................... 651
in tcsetattr( ) ............................................... 654
in tcgetpgrp( ) ............................................... 655
in write( ) ..................................................... 741
EBADMSG ..................................................... 26, 750
in read( ) ........................................................ 461
EBUSY .......................................................... 26, 750
in rename( ) .................................................... 490
in rmdir( ) ...................................................... 496
in unlink( ) ..................................................... 689
ECHILD .......................................................... 26, 750
in pclose( ) ..................................................... 433
in wait() ......................................................... 702
ECHO ............................................................. 849
ECHOE ........................................................... 849
ECHOK ........................................................... 849
ECHONL ......................................................... 849
ECONNABORTED ............................................. 26, 750
ECONNREFUSED ............................................. 26, 750
ECONNRESET ................................................ 26, 750
evt() ............................................................. 120
EDEADLK ........................................................ 26, 750
ef() ............................................................... 151
EDESTADDRREQ ............................................. 26, 750
EDOM ............................................................ 26, 750
in acos( ) ........................................................ 45
in asin( ) ........................................................ 51
in atan2( ) ...................................................... 55
in erf( ) ........................................................ 129
Index

in exp() .................................................................138
in fabs() ...............................................................140
in hypot() .............................................................293
in j0() ................................................................342
in ldexp() ..............................................................351
in lgamma() ...........................................................354
in log() ................................................................365
in log10() ..............................................................366
in pow() ................................................................440
in sin() ..................................................................577
in sqrt() ..................................................................582
in y0() ....................................................................744
EDQUOT .................................................................26, 750
EEXIST .................................................................26, 750
in link() .................................................................355
in mkdir() ..............................................................388
in mkfifo() ............................................................390
in open() .................................................................423
in shmget() ..........................................................543
EFAULT .................................................................26, 750
EFBIG .................................................................27, 750
in fclose() .............................................................146
in fflush() .............................................................159
in fputc() ..............................................................190
in fputwc() ...........................................................193
in fseek() ..............................................................207
in write() ..............................................................741
EHOSTUNREACH ....................................................27, 750
EI .................................................................11
in strftime() ..........................................................601
in strptime() ..........................................................614
in wcscoll() ...........................................................711
in wcstombs() ........................................................714
in wcstotime() ........................................................732
EIDRM .................................................................27, 750
EISOCK .................................................................27, 750
in fprintf() ............................................................188
in fscanf() .............................................................204
in mbmblen() ..........................................................379
in mbstowcs() ........................................................380
in mbtowc() ...........................................................381
in ungets() ............................................................688
in wcstombs() ........................................................727
EINPROGRESS .......................................................27, 750
EINTR .................................................................27, 750
in(close) .................................................................72
in catgets() ...........................................................73
in close() ..............................................................93
in dup() .................................................................118
in endgrent() .........................................................123
in fclose() ............................................................146
in fcntl() ...............................................................150
in fflush() .............................................................159
in fgetc() ..............................................................162
in fgetwc() ............................................................166
in fopen() ..............................................................179
in fputc() ..............................................................190
in fputwc() ...........................................................193
in freopen() ...........................................................198
in fseek() ..............................................................207
in fsync() ..............................................................214
in getgrent() ........................................................240
in getgrgid() ........................................................241
in getgrnam() ........................................................241
in getpid() ............................................................255
in getpwnam() ........................................................265
in getpwuid() ........................................................267
in open() ...............................................................423
in pause() .............................................................432
in read() ...............................................................461
in sigsuspend() ......................................................576
in tcdrain() ...........................................................642
in tcssetattr() ........................................................654
in tmpfile() ...........................................................663
in wait() ..............................................................702
in write() .............................................................741
EINVAL .............................................................150
in sigaction() ........................................................550
in shmget() ..........................................................543
in shmdt() ............................................................542
in shmat() ............................................................538
in setgid() ............................................................490
in sigaction() ........................................................516
in getcwd() ...........................................................229
in getgroups() .......................................................242
in kill() ...............................................................344
in lseek() .............................................................375
in pathconf() ........................................................430
in open() ..............................................................438
in read() .............................................................461
in rename() ...........................................................490
in setgid() ............................................................516
in setpgid() ..........................................................526
in setuid() ............................................................535
in shmat() ............................................................538
in shmctl() ...........................................................540
in shmdt() ............................................................542
in shmget() ..........................................................543
in sigaction() ........................................................550
Index

EISCONN ........................................................... 27, 750
EISDIR ............................................................... 27, 750
in fopen() ............................................................ 179
in freopen() ........................................................ 198
in open() .............................................................. 423
in read() .............................................................. 461

EIO .................................................................. 27, 750
in close() ............................................................. 93
in fclose() ........................................................... 146
in fflush() ............................................................ 159
in fgetc() ............................................................. 162
in fgetwc() ........................................................... 166
in fputc() .............................................................. 190
in fputwc() ........................................................... 193
in fseek() ............................................................. 207
in fstat() .............................................................. 210
in fsync() ............................................................. 214
in getpass() .......................................................... 255
in lstat() ............................................................. 376
in open() .............................................................. 423
in read() .............................................................. 461
in rename() .......................................................... 490
in rdmdir() .......................................................... 496
in stat() .............................................................. 587
in tcdrain() .......................................................... 642
in tcf10w() ............................................................ 644
in tcf10w() ............................................................. 646
in tcsendbreak() .................................................... 651
in tcsetattr() ........................................................ 654
in write() ............................................................. 741

ELOOP .............................................................. 27, 750
in access() ........................................................... 43
in chmod() ............................................................ 84
in chown() ........................................................... 86
in chroot() ............................................................ 88
in exec ............................................................... 133
in fopen() ............................................................. 179
in freopen() ........................................................ 198
in ftw() ............................................................... 221
in link() .............................................................. 355
in mkdir() ............................................................ 388
in mkfifo() ........................................................... 390
in nftw() ............................................................. 417
in open() .............................................................. 423
in opendir() ........................................................ 425
in pathconf() ....................................................... 430
in rename() .......................................................... 490
in rmdir() ........................................................... 496
in stat() .............................................................. 587
in unlink() .......................................................... 689

EMFILE ........................................................... 27, 750
in catopen() ........................................................ 74
in dup() .............................................................. 118
in fcntl() ............................................................. 150
in fdopen() ........................................................ 155
in fopen() ............................................................ 179
in freopen() ........................................................ 198
in getgrgid() ....................................................... 240
in getgrnam() ..................................................... 241
in getpass() ........................................................ 255
in getpwnam() ..................................................... 265
in getpwuid() ...................................................... 267
in open() .............................................................. 423
in opendir() ........................................................ 425
in pipe() ............................................................. 435
in popen() ........................................................... 438
in shmat() ........................................................... 538
in tmpfile() .......................................................... 663

EMLINK ........................................................... 27, 750
in link() .............................................................. 355
in mkdir() ............................................................ 388
in rename() .......................................................... 490

EMSGSIZE ......................................................... 27, 750

EMULTHOP ........................................................ 27, 750

ENAMETOOLONG .............................................. 28, 750
in access() ........................................................... 43
in catopen() ........................................................ 74
in chmod() ............................................................ 82
in chdir() ............................................................ 82
in chown() ........................................................... 84
## Index

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>in chown()</td>
<td>86</td>
</tr>
<tr>
<td>in chroot()</td>
<td>88</td>
</tr>
<tr>
<td>in exec</td>
<td>133</td>
</tr>
<tr>
<td>in fopen()</td>
<td>179</td>
</tr>
<tr>
<td>in freopen()</td>
<td>198</td>
</tr>
<tr>
<td>in ftw()</td>
<td>221</td>
</tr>
<tr>
<td>in link()</td>
<td>355</td>
</tr>
<tr>
<td>in mkdir()</td>
<td>388</td>
</tr>
<tr>
<td>in mkfifo()</td>
<td>390</td>
</tr>
<tr>
<td>in open()</td>
<td>423</td>
</tr>
<tr>
<td>in opendir()</td>
<td>425</td>
</tr>
<tr>
<td>in pathconf()</td>
<td>430</td>
</tr>
<tr>
<td>in rename()</td>
<td>490</td>
</tr>
<tr>
<td>in rmdir()</td>
<td>496</td>
</tr>
<tr>
<td>in stat()</td>
<td>587</td>
</tr>
<tr>
<td>in unlink()</td>
<td>689</td>
</tr>
<tr>
<td>in utime()</td>
<td>693</td>
</tr>
<tr>
<td>in utimes()</td>
<td>695</td>
</tr>
<tr>
<td>encrypt()</td>
<td>122</td>
</tr>
<tr>
<td>endgrent()</td>
<td>123</td>
</tr>
<tr>
<td>endpwent()</td>
<td>124</td>
</tr>
<tr>
<td>endutent()</td>
<td>125</td>
</tr>
<tr>
<td>ENETDOWN</td>
<td>28,750</td>
</tr>
<tr>
<td>ENETUNREACH</td>
<td>28,750</td>
</tr>
<tr>
<td>ENFILE</td>
<td>28,750</td>
</tr>
<tr>
<td>in catopen()</td>
<td>74</td>
</tr>
<tr>
<td>in fopen()</td>
<td>179</td>
</tr>
<tr>
<td>in freopen()</td>
<td>198</td>
</tr>
<tr>
<td>in getgrgid()</td>
<td>240</td>
</tr>
<tr>
<td>in getgname()</td>
<td>241</td>
</tr>
<tr>
<td>in getlogin()</td>
<td>246</td>
</tr>
<tr>
<td>in getpass()</td>
<td>255</td>
</tr>
<tr>
<td>in getpwnam()</td>
<td>265</td>
</tr>
<tr>
<td>in getpwuid()</td>
<td>267</td>
</tr>
<tr>
<td>in open()</td>
<td>423</td>
</tr>
<tr>
<td>in opendir()</td>
<td>425</td>
</tr>
<tr>
<td>in pipe()</td>
<td>435</td>
</tr>
<tr>
<td>in tmpfile()</td>
<td>663</td>
</tr>
<tr>
<td>ENOBIFS</td>
<td>28,750</td>
</tr>
<tr>
<td>ENODATA</td>
<td>28,750</td>
</tr>
<tr>
<td>ENODEV</td>
<td>28,750</td>
</tr>
<tr>
<td>ENOENT</td>
<td>28,750</td>
</tr>
<tr>
<td>in access()</td>
<td>43</td>
</tr>
<tr>
<td>in catopen()</td>
<td>74</td>
</tr>
<tr>
<td>in chdir()</td>
<td>82</td>
</tr>
<tr>
<td>in chmod()</td>
<td>84</td>
</tr>
<tr>
<td>in chown()</td>
<td>86</td>
</tr>
<tr>
<td>in chroot()</td>
<td>88</td>
</tr>
<tr>
<td>in exec</td>
<td>134</td>
</tr>
<tr>
<td>in fopen()</td>
<td>179</td>
</tr>
<tr>
<td>in freopen()</td>
<td>198</td>
</tr>
<tr>
<td>in ftw()</td>
<td>221</td>
</tr>
<tr>
<td>in link()</td>
<td>355</td>
</tr>
<tr>
<td>in lstat()</td>
<td>376</td>
</tr>
<tr>
<td>in mkdir()</td>
<td>388</td>
</tr>
<tr>
<td>in mkfifo()</td>
<td>390</td>
</tr>
<tr>
<td>in open()</td>
<td>423</td>
</tr>
<tr>
<td>in opendir()</td>
<td>425</td>
</tr>
<tr>
<td>in pathconf()</td>
<td>430</td>
</tr>
<tr>
<td>in readdir()</td>
<td>463</td>
</tr>
<tr>
<td>in rename()</td>
<td>490</td>
</tr>
<tr>
<td>in rmdir()</td>
<td>496</td>
</tr>
<tr>
<td>in shmget()</td>
<td>543</td>
</tr>
<tr>
<td>in stat()</td>
<td>587</td>
</tr>
<tr>
<td>in unlink()</td>
<td>689</td>
</tr>
<tr>
<td>in utime()</td>
<td>693</td>
</tr>
<tr>
<td>in utimes()</td>
<td>695</td>
</tr>
<tr>
<td>ENOEXEC</td>
<td>28,751</td>
</tr>
<tr>
<td>in exec</td>
<td>134</td>
</tr>
<tr>
<td>ENOLCK</td>
<td>28,751</td>
</tr>
<tr>
<td>in fcntl()</td>
<td>150</td>
</tr>
<tr>
<td>ENOLINK</td>
<td>28,751</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>28,751</td>
</tr>
<tr>
<td>in calloc()</td>
<td>71</td>
</tr>
<tr>
<td>in catopen()</td>
<td>74</td>
</tr>
<tr>
<td>in exec</td>
<td>134</td>
</tr>
<tr>
<td>in fdopen()</td>
<td>155</td>
</tr>
<tr>
<td>in fgetwc()</td>
<td>166</td>
</tr>
<tr>
<td>in fopen()</td>
<td>179</td>
</tr>
<tr>
<td>in fork()</td>
<td>181</td>
</tr>
<tr>
<td>in fputwc()</td>
<td>194</td>
</tr>
<tr>
<td>in freopen()</td>
<td>199</td>
</tr>
<tr>
<td>in hsearch()</td>
<td>291</td>
</tr>
<tr>
<td>in open()</td>
<td>423</td>
</tr>
<tr>
<td>in putenv()</td>
<td>446</td>
</tr>
<tr>
<td>in shmat()</td>
<td>538</td>
</tr>
<tr>
<td>in shmget()</td>
<td>543</td>
</tr>
<tr>
<td>in tempsnam()</td>
<td>658</td>
</tr>
<tr>
<td>in tmpfile()</td>
<td>663</td>
</tr>
<tr>
<td>ENOMSG</td>
<td>28,751</td>
</tr>
<tr>
<td>in catgets()</td>
<td>73</td>
</tr>
<tr>
<td>ENOPROTOOPT</td>
<td>28,751</td>
</tr>
<tr>
<td>ENOSPC</td>
<td>28,751</td>
</tr>
<tr>
<td>in fclose()</td>
<td>146</td>
</tr>
<tr>
<td>in ffflush()</td>
<td>159</td>
</tr>
<tr>
<td>in fopen()</td>
<td>179</td>
</tr>
<tr>
<td>in fputc()</td>
<td>190</td>
</tr>
<tr>
<td>in fputwc()</td>
<td>193</td>
</tr>
<tr>
<td>in freopen()</td>
<td>198</td>
</tr>
<tr>
<td>in fseek()</td>
<td>208</td>
</tr>
<tr>
<td>in link()</td>
<td>355</td>
</tr>
<tr>
<td>in mkdir()</td>
<td>388</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------</td>
</tr>
<tr>
<td>in mkfifo( )</td>
<td>390</td>
</tr>
<tr>
<td>in open( )</td>
<td>423</td>
</tr>
<tr>
<td>in rename( )</td>
<td>490</td>
</tr>
<tr>
<td>in shmget( )</td>
<td>543</td>
</tr>
<tr>
<td>in tmpfile( )</td>
<td>663</td>
</tr>
<tr>
<td>in write( )</td>
<td>741</td>
</tr>
<tr>
<td>ENOTCONN</td>
<td>29, 751</td>
</tr>
<tr>
<td>in open( )</td>
<td>423</td>
</tr>
<tr>
<td>ENOSTR</td>
<td>29, 751</td>
</tr>
<tr>
<td>in crypt( )</td>
<td>104</td>
</tr>
<tr>
<td>in encrypt( )</td>
<td>122</td>
</tr>
<tr>
<td>in setkey( )</td>
<td>522</td>
</tr>
<tr>
<td>in shmat( )</td>
<td>538</td>
</tr>
<tr>
<td>in shmtl( )</td>
<td>540</td>
</tr>
<tr>
<td>in shmdt( )</td>
<td>542</td>
</tr>
<tr>
<td>in shmemt( )</td>
<td>544</td>
</tr>
<tr>
<td>in strftime( )</td>
<td>603</td>
</tr>
<tr>
<td>in strptime( )</td>
<td>616</td>
</tr>
<tr>
<td>in wcscoll( )</td>
<td>711</td>
</tr>
<tr>
<td>in westtime( )</td>
<td>714</td>
</tr>
<tr>
<td>in wcsxfrm( )</td>
<td>732</td>
</tr>
<tr>
<td>ENOTCONN</td>
<td>29, 751</td>
</tr>
<tr>
<td>ENOTDIR</td>
<td>29, 751</td>
</tr>
<tr>
<td>in access( )</td>
<td>43</td>
</tr>
<tr>
<td>in catopen( )</td>
<td>75</td>
</tr>
<tr>
<td>in chdir( )</td>
<td>82</td>
</tr>
<tr>
<td>in chmod( )</td>
<td>84</td>
</tr>
<tr>
<td>in chown( )</td>
<td>86</td>
</tr>
<tr>
<td>in chroot( )</td>
<td>88</td>
</tr>
<tr>
<td>in exec</td>
<td>134</td>
</tr>
<tr>
<td>in fopen( )</td>
<td>179</td>
</tr>
<tr>
<td>in freen( )</td>
<td>198</td>
</tr>
<tr>
<td>in ftw( )</td>
<td>221</td>
</tr>
<tr>
<td>in link( )</td>
<td>355</td>
</tr>
<tr>
<td>in mkdir( )</td>
<td>388</td>
</tr>
<tr>
<td>in mkfifo( )</td>
<td>390</td>
</tr>
<tr>
<td>in open( )</td>
<td>423</td>
</tr>
<tr>
<td>in opendir( )</td>
<td>425</td>
</tr>
<tr>
<td>in pathconf( )</td>
<td>430</td>
</tr>
<tr>
<td>in rename( )</td>
<td>490</td>
</tr>
<tr>
<td>in rmdir( )</td>
<td>496</td>
</tr>
<tr>
<td>in stat( )</td>
<td>587</td>
</tr>
<tr>
<td>in unlink( )</td>
<td>689</td>
</tr>
<tr>
<td>in utime( )</td>
<td>693</td>
</tr>
<tr>
<td>in utimes( )</td>
<td>695</td>
</tr>
<tr>
<td>ENOTEMPTY</td>
<td>29, 751</td>
</tr>
<tr>
<td>ENOTSOCK</td>
<td>29, 751</td>
</tr>
<tr>
<td>ENOTTY</td>
<td>29, 751</td>
</tr>
<tr>
<td>in isatty( )</td>
<td>319</td>
</tr>
<tr>
<td>in tcdrain( )</td>
<td>642</td>
</tr>
<tr>
<td>in tcflow( )</td>
<td>644</td>
</tr>
<tr>
<td>in tcflush( )</td>
<td>646</td>
</tr>
<tr>
<td>in tcgetpgrp( )</td>
<td>649</td>
</tr>
<tr>
<td>in tcsendbreak( )</td>
<td>651</td>
</tr>
<tr>
<td>in tcsetattr( )</td>
<td>654</td>
</tr>
<tr>
<td>in tcsetpgrp( )</td>
<td>655</td>
</tr>
<tr>
<td>environ</td>
<td>127</td>
</tr>
<tr>
<td>ENXIO</td>
<td>29, 751</td>
</tr>
<tr>
<td>in fgetwc( )</td>
<td>166</td>
</tr>
<tr>
<td>in fopen( )</td>
<td>179</td>
</tr>
<tr>
<td>in fputc( )</td>
<td>190</td>
</tr>
<tr>
<td>in fputwc( )</td>
<td>194</td>
</tr>
<tr>
<td>in freopen( )</td>
<td>198-199</td>
</tr>
<tr>
<td>in getlogin( )</td>
<td>246</td>
</tr>
<tr>
<td>in getpass( )</td>
<td>255</td>
</tr>
<tr>
<td>in open( )</td>
<td>423</td>
</tr>
<tr>
<td>in XCU specification</td>
<td>208</td>
</tr>
<tr>
<td>EACCES</td>
<td>29, 751</td>
</tr>
<tr>
<td>in chmod( )</td>
<td>84</td>
</tr>
<tr>
<td>in chown( )</td>
<td>86</td>
</tr>
<tr>
<td>in chroot( )</td>
<td>88</td>
</tr>
<tr>
<td>in msgctl( )</td>
<td>404</td>
</tr>
<tr>
<td>in nice( )</td>
<td>418</td>
</tr>
<tr>
<td>in rename( )</td>
<td>490</td>
</tr>
<tr>
<td>in rmdir( )</td>
<td>496</td>
</tr>
<tr>
<td>in semctl( )</td>
<td>507</td>
</tr>
<tr>
<td>in setgid( )</td>
<td>516</td>
</tr>
<tr>
<td>in setpgid( )</td>
<td>526</td>
</tr>
<tr>
<td>in setuid( )</td>
<td>533</td>
</tr>
<tr>
<td>in setgid( )</td>
<td>535</td>
</tr>
<tr>
<td>in shmtl( )</td>
<td>540</td>
</tr>
<tr>
<td>in tcsetpgrp( )</td>
<td>655</td>
</tr>
<tr>
<td>in ulimit()</td>
<td>684</td>
</tr>
<tr>
<td>in unlink( )</td>
<td>689</td>
</tr>
<tr>
<td>in utime( )</td>
<td>693</td>
</tr>
<tr>
<td>in utimes( )</td>
<td>695</td>
</tr>
<tr>
<td>in XCU specification</td>
<td>355</td>
</tr>
<tr>
<td>Epipe</td>
<td>29, 751</td>
</tr>
<tr>
<td>in fclose( )</td>
<td>146</td>
</tr>
<tr>
<td>in fflush( )</td>
<td>159</td>
</tr>
<tr>
<td>in fputc( )</td>
<td>190</td>
</tr>
<tr>
<td>in fputwc( )</td>
<td>193</td>
</tr>
<tr>
<td>in fseek( )</td>
<td>208</td>
</tr>
<tr>
<td>in write( )</td>
<td>741</td>
</tr>
<tr>
<td>EPROM</td>
<td>30, 751</td>
</tr>
<tr>
<td>in protosupport( )</td>
<td>30, 751</td>
</tr>
<tr>
<td>EPROTONOSUPPORT</td>
<td>30, 751</td>
</tr>
<tr>
<td>EPROTOTYPE</td>
<td>30, 751</td>
</tr>
</tbody>
</table>
Index

ERA ................................................................. 766
erand48() ....................................................... 116, 128
ERANGE ......................................................... 30, 751
  in asin() ......................................................... 51
  in atan() ......................................................... 54
  in atan2() ....................................................... 55
  in ceil() .......................................................... 77
  in cos() ........................................................... 101
  in cosh() ........................................................ 102
  in erf() ......................................................... 129
  in exp() ........................................................... 138
  in fabs() ......................................................... 140
  in floor() ........................................................ 170
  in fmod() ........................................................ 172
  in getcwd() ..................................................... 229
  in hypot() ....................................................... 293
  in j0() .............................................................. 342
  in ldexp() ...................................................... 351
  in lgamma() .................................................... 354
  in log() .......................................................... 365
  in log10() ........................................................ 366
  in modf() ....................................................... 401
  in pow() .......................................................... 440
  in sin() ........................................................... 577
  in sinh() ........................................................ 578
  in strtod() ...................................................... 621
  in strtol() ...................................................... 624
  in strtoul() .................................................... 626
  in tan() ........................................................... 640
  in tanh() ........................................................ 641
  in wctod() ..................................................... 722
  in wcstol() ..................................................... 726
  in wcstoul() .................................................... 729
  in write() ....................................................... 742
  in y0() ............................................................ 744
ERA_D_FMT ..................................................... 766
ERA_D_T_FMT ................................................... 766
ERA_T_FMT ..................................................... 766
erf() ............................................................. 129
EROFS ........................................................... 30, 751
  in access() ..................................................... 43
  in chmod() ..................................................... 84
  in chown() ..................................................... 86
  in fopen() ..................................................... 179
  in freopen() .................................................. 198
  in link() ......................................................... 355
  in mkdir() ...................................................... 388
  in mkfifo() ..................................................... 390
  in open() ....................................................... 423
  in rename() ................................................... 490
  in rmdir() ..................................................... 497
  in unlink() .................................................... 690
  in utime() ..................................................... 693
  in utimes() .................................................... 695
erro ............................................................ 130
error numbers ................................................ 25
  additional ..................................................... 31
ESPIPE ......................................................... 30, 751
  in fgetpos() .................................................. 164
  in fsetpos() .................................................. 209
  in ftell() ....................................................... 215
  in fseek() ...................................................... 375
ESRCH ......................................................... 30, 751
  in kill() ......................................................... 344
  in setpgid() .................................................. 526
ESTALE ......................................................... 30, 751
ETIME .......................................................... 30, 751
ETIMEOUT .................................................... 30, 751
ETXTBSY ....................................................... 30, 751
  in access() ..................................................... 43
  in freopen() .................................................. 199
  in rename() ................................................... 491
  in unlink() .................................................... 690
EWOULDBLOCK ............................................... 30, 751
EX ............................................................. 11, 19-20, 27-28, 30, 37
  FIPS ............................................................ 11
    in <cpio.h> .................................................. 747
    in <ctype.h> ................................................. 748
    in <dirent.h> ................................................. 749
    in <errno.h> .................................................. 750-751
    in <errno.h> (FIPS) ........................................... 751
    in <fcntl.h> .................................................. 753-754
    in <ftw.h> .................................................... 760
    in <grp.h> .................................................... 763
    in <iconv.h> .................................................. 764
    in <langinfo.h> ............................................. 765
    in <limits.h> ................................................ 770, 775-776
      in <limits.h> (FIPS) ....................................... 770, 777
      in <math.h> ................................................ 780-781
      in <monetary.h> .......................................... 783
      in <netinet.h> ............................................. 785
      in <pwd.h> .................................................. 787
      in <regexp.h> .............................................. 791
      in <search.h> .............................................. 792
      in <signal.h> .............................................. 794-795
      in <signal.h> (FIPS) ....................................... 795
      in <stdio.h> ................................................. 804-806
      in <stdlib.h> .............................................. 807-808
      in <string.h> .............................................. 810
      in <sys/IPC.h> ............................................. 820
      in <sys/msg.h> ............................................. 823
      in <sys/sem.h> ............................................. 827
### Index

<table>
<thead>
<tr>
<th>Function</th>
<th>Page Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>in <code>&lt;sys/shm.h&gt;</code></td>
<td>829</td>
</tr>
<tr>
<td>in <code>&lt;sys/stat.h&gt;</code></td>
<td>830</td>
</tr>
<tr>
<td>in <code>&lt;sys/types.h&gt;</code></td>
<td>838</td>
</tr>
<tr>
<td>in <code>&lt;sys/wait.h&gt;</code></td>
<td>842</td>
</tr>
<tr>
<td>in <code>&lt;termios.h&gt;</code></td>
<td>847, 849</td>
</tr>
<tr>
<td>in <code>&lt;time.h&gt;</code></td>
<td>851-852</td>
</tr>
<tr>
<td>in <code>&lt;unistd.h&gt;</code> (FIPS)</td>
<td>854</td>
</tr>
<tr>
<td>in <code>&lt;unistd.h&gt;</code></td>
<td>855-860</td>
</tr>
<tr>
<td>in <code>&lt;unistd.h&gt;</code> (FIPS)</td>
<td>856</td>
</tr>
<tr>
<td>in <code>&lt;utime.h&gt;</code></td>
<td>863</td>
</tr>
<tr>
<td>in <code>&lt;varargs.h&gt;</code></td>
<td>865</td>
</tr>
<tr>
<td>in <code>&lt;wchar.h&gt;</code></td>
<td>867</td>
</tr>
<tr>
<td>in <code>access()</code></td>
<td>43</td>
</tr>
<tr>
<td>in <code>access()</code> (FIPS)</td>
<td>43</td>
</tr>
<tr>
<td>in <code>acos()</code></td>
<td>45</td>
</tr>
<tr>
<td>in <code>advance()</code></td>
<td>47</td>
</tr>
<tr>
<td>in <code>asin()</code></td>
<td>51</td>
</tr>
<tr>
<td>in <code>atan()</code></td>
<td>54</td>
</tr>
<tr>
<td>in <code>atan2()</code></td>
<td>55</td>
</tr>
<tr>
<td>in <code>calloc()</code></td>
<td>71</td>
</tr>
<tr>
<td>in <code>catclose()</code></td>
<td>72</td>
</tr>
<tr>
<td>in <code>catgets()</code></td>
<td>73</td>
</tr>
<tr>
<td>in <code>catopen()</code></td>
<td>74</td>
</tr>
<tr>
<td>in <code>ceil()</code></td>
<td>77</td>
</tr>
<tr>
<td>in <code>cfsetispeed()</code></td>
<td>80</td>
</tr>
<tr>
<td>in <code>cfsetispeed()</code></td>
<td>81</td>
</tr>
<tr>
<td>in <code>chdir()</code> (FIPS)</td>
<td>82</td>
</tr>
<tr>
<td>in <code>chmod()</code> (FIPS)</td>
<td>85</td>
</tr>
<tr>
<td>in <code>chmod()</code></td>
<td>84</td>
</tr>
<tr>
<td>in <code>chown()</code> (FIPS)</td>
<td>86</td>
</tr>
<tr>
<td>in <code>chown()</code></td>
<td>86</td>
</tr>
<tr>
<td>in <code>chroot()</code></td>
<td>88</td>
</tr>
<tr>
<td>in <code>clock()</code></td>
<td>91</td>
</tr>
<tr>
<td>in <code>clockdir()</code></td>
<td>94</td>
</tr>
<tr>
<td>in <code>compile()</code></td>
<td>98</td>
</tr>
<tr>
<td>in <code>cos()</code></td>
<td>101</td>
</tr>
<tr>
<td>in <code>cosh()</code></td>
<td>102</td>
</tr>
<tr>
<td>in <code>crypt()</code></td>
<td>104</td>
</tr>
<tr>
<td>in <code>csuirid()</code></td>
<td>107</td>
</tr>
<tr>
<td>in <code>daylight()</code></td>
<td>109</td>
</tr>
<tr>
<td>in <code>drand48()</code></td>
<td>116</td>
</tr>
<tr>
<td>in <code>encrypt()</code></td>
<td>122</td>
</tr>
<tr>
<td>in <code>erand48()</code></td>
<td>128</td>
</tr>
<tr>
<td>in <code>erf()</code></td>
<td>129</td>
</tr>
<tr>
<td>in <code>errno</code></td>
<td>130</td>
</tr>
<tr>
<td>in <code>exec()</code></td>
<td>131-134</td>
</tr>
<tr>
<td>in <code>exec</code> (FIPS)</td>
<td>132-133</td>
</tr>
<tr>
<td>in <code>exit()</code></td>
<td>136</td>
</tr>
<tr>
<td>in <code>exp()</code></td>
<td>138</td>
</tr>
<tr>
<td>in <code>fabs()</code></td>
<td>140</td>
</tr>
<tr>
<td>in <code>fclose()</code></td>
<td>146</td>
</tr>
<tr>
<td>in <code>fcntl()</code></td>
<td>149-150</td>
</tr>
<tr>
<td>in <code>fdoopen()</code></td>
<td>155</td>
</tr>
<tr>
<td>in <code>fflush()</code></td>
<td>159</td>
</tr>
<tr>
<td>in <code>fgetc()</code></td>
<td>162</td>
</tr>
<tr>
<td>in <code>fgetpos()</code></td>
<td>164</td>
</tr>
<tr>
<td>in <code>fileno()</code></td>
<td>169</td>
</tr>
<tr>
<td>in <code>floor()</code></td>
<td>170</td>
</tr>
<tr>
<td>in <code>fmod()</code></td>
<td>172</td>
</tr>
<tr>
<td>in <code>fopen()</code></td>
<td>179</td>
</tr>
<tr>
<td>in <code>fopen</code> (FIPS)</td>
<td>179</td>
</tr>
<tr>
<td>in <code>fork()</code></td>
<td>181</td>
</tr>
<tr>
<td>in <code>fprintf()</code></td>
<td>184-188</td>
</tr>
<tr>
<td>in <code>fputc()</code></td>
<td>190</td>
</tr>
<tr>
<td>in <code>freopen()</code></td>
<td>198</td>
</tr>
<tr>
<td>in <code>freopen</code> (FIPS)</td>
<td>198</td>
</tr>
<tr>
<td>in <code>frexp()</code></td>
<td>200</td>
</tr>
<tr>
<td>in <code>fscanf()</code></td>
<td>201-204</td>
</tr>
<tr>
<td>in <code>fseek()</code></td>
<td>207-208</td>
</tr>
<tr>
<td>in <code>fsetpos()</code></td>
<td>209</td>
</tr>
<tr>
<td>in <code>fsync()</code></td>
<td>214</td>
</tr>
<tr>
<td>in <code>ftw()</code></td>
<td>220</td>
</tr>
<tr>
<td>in <code>gamma()</code></td>
<td>224</td>
</tr>
<tr>
<td>in <code>getcwd()</code></td>
<td>229</td>
</tr>
<tr>
<td>in <code>getenv()</code></td>
<td>236</td>
</tr>
<tr>
<td>in <code>getgrent()</code></td>
<td>240</td>
</tr>
<tr>
<td>in <code>getgrent()</code></td>
<td>241</td>
</tr>
<tr>
<td>in <code>getlogin()</code></td>
<td>246</td>
</tr>
<tr>
<td>in <code>getpwnam()</code></td>
<td>255</td>
</tr>
<tr>
<td>in <code>getpwuid()</code></td>
<td>265</td>
</tr>
<tr>
<td>in <code>getpwuid()</code></td>
<td>267</td>
</tr>
<tr>
<td>in <code>getw()</code></td>
<td>278</td>
</tr>
<tr>
<td>in <code>hcreate()</code></td>
<td>288</td>
</tr>
<tr>
<td>in <code>hdestroy()</code></td>
<td>289</td>
</tr>
<tr>
<td>in <code>hsearch()</code></td>
<td>290</td>
</tr>
<tr>
<td>in <code>hypot()</code></td>
<td>293</td>
</tr>
<tr>
<td>in <code>iconv()</code></td>
<td>294</td>
</tr>
<tr>
<td>in <code>iconv_close()</code></td>
<td>296</td>
</tr>
<tr>
<td>in <code>iconv_open()</code></td>
<td>297</td>
</tr>
<tr>
<td>in <code>isascii()</code></td>
<td>317</td>
</tr>
<tr>
<td>in <code>isatty()</code></td>
<td>319</td>
</tr>
<tr>
<td>in <code>isnan()</code></td>
<td>324</td>
</tr>
<tr>
<td>in <code>j0()</code></td>
<td>342</td>
</tr>
<tr>
<td>in <code>jrand48()</code></td>
<td>343</td>
</tr>
<tr>
<td>in <code>kill()</code></td>
<td>344</td>
</tr>
<tr>
<td>in <code>kill</code> (FIPS)</td>
<td>344</td>
</tr>
<tr>
<td>in <code>lcmag48()</code></td>
<td>350</td>
</tr>
<tr>
<td>in <code>ldexp()</code></td>
<td>351</td>
</tr>
<tr>
<td>in <code>lfind()</code></td>
<td>353</td>
</tr>
<tr>
<td>in <code>lgamma()</code></td>
<td>354</td>
</tr>
<tr>
<td>in <code>link()</code> (FIPS)</td>
<td>355</td>
</tr>
<tr>
<td>in <code>loc1</code></td>
<td>357</td>
</tr>
</tbody>
</table>
### Index

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>in localeconv()</td>
<td>359</td>
</tr>
<tr>
<td>in locs</td>
<td>364</td>
</tr>
<tr>
<td>in log()</td>
<td>365</td>
</tr>
<tr>
<td>in log10()</td>
<td>366</td>
</tr>
<tr>
<td>in lrand48()</td>
<td>372</td>
</tr>
<tr>
<td>in lsearch()</td>
<td>373</td>
</tr>
<tr>
<td>in malloc()</td>
<td>378</td>
</tr>
<tr>
<td>in mblen()</td>
<td>379</td>
</tr>
<tr>
<td>in mbstowcs()</td>
<td>380</td>
</tr>
<tr>
<td>in mbtowc()</td>
<td>381</td>
</tr>
<tr>
<td>in memcpy()</td>
<td>382</td>
</tr>
<tr>
<td>in mkdir() (FIPS)</td>
<td>388</td>
</tr>
<tr>
<td>in mkfifo() (FIPS)</td>
<td>390</td>
</tr>
<tr>
<td>in modf()</td>
<td>401</td>
</tr>
<tr>
<td>in mrand48()</td>
<td>403</td>
</tr>
<tr>
<td>in msgctl()</td>
<td>404</td>
</tr>
<tr>
<td>in msgget()</td>
<td>406</td>
</tr>
<tr>
<td>in msgsnd()</td>
<td>408</td>
</tr>
<tr>
<td>in msgsnd()</td>
<td>410</td>
</tr>
<tr>
<td>in nice()</td>
<td>418</td>
</tr>
<tr>
<td>in nl_langinfo()</td>
<td>419</td>
</tr>
<tr>
<td>in nrand48()</td>
<td>420</td>
</tr>
<tr>
<td>in open()</td>
<td>422-424</td>
</tr>
<tr>
<td>in open() (FIPS)</td>
<td>421, 423</td>
</tr>
<tr>
<td>in opendir() (FIPS)</td>
<td>425</td>
</tr>
<tr>
<td>in pathconf() (FIPS)</td>
<td>430</td>
</tr>
<tr>
<td>in popen()</td>
<td>438</td>
</tr>
<tr>
<td>in pow()</td>
<td>440</td>
</tr>
<tr>
<td>in putenv()</td>
<td>446</td>
</tr>
<tr>
<td>in putw()</td>
<td>451</td>
</tr>
<tr>
<td>in raise()</td>
<td>455</td>
</tr>
<tr>
<td>in rand()</td>
<td>456</td>
</tr>
<tr>
<td>in read()</td>
<td>461</td>
</tr>
<tr>
<td>in read() (FIPS)</td>
<td>460</td>
</tr>
<tr>
<td>in readdir() (FIPS)</td>
<td>463</td>
</tr>
<tr>
<td>in realloca()</td>
<td>467</td>
</tr>
<tr>
<td>in regexp()</td>
<td>480</td>
</tr>
<tr>
<td>in remove()</td>
<td>487</td>
</tr>
<tr>
<td>in rename()</td>
<td>491</td>
</tr>
<tr>
<td>in rename() (FIPS)</td>
<td>490</td>
</tr>
<tr>
<td>in rewinddir()</td>
<td>493</td>
</tr>
<tr>
<td>in rmdir()</td>
<td>496</td>
</tr>
<tr>
<td>in seed48()</td>
<td>501</td>
</tr>
<tr>
<td>in seekdir()</td>
<td>502</td>
</tr>
<tr>
<td>in semctl()</td>
<td>506</td>
</tr>
<tr>
<td>in semget()</td>
<td>509</td>
</tr>
<tr>
<td>in semop()</td>
<td>511</td>
</tr>
<tr>
<td>in setgid() (FIPS)</td>
<td>516</td>
</tr>
<tr>
<td>in setkey() (FIPS)</td>
<td>522</td>
</tr>
<tr>
<td>in setlocale()</td>
<td>523</td>
</tr>
<tr>
<td>in setuid() (FIPS)</td>
<td>535</td>
</tr>
<tr>
<td>in setvbuf()</td>
<td>537</td>
</tr>
<tr>
<td>in shmat()</td>
<td>538</td>
</tr>
<tr>
<td>in shmct1()</td>
<td>540</td>
</tr>
<tr>
<td>in shmdt()</td>
<td>542</td>
</tr>
<tr>
<td>in shmmget()</td>
<td>543</td>
</tr>
<tr>
<td>in sigaction()</td>
<td>549</td>
</tr>
<tr>
<td>in signgam()</td>
<td>566</td>
</tr>
<tr>
<td>in sin()</td>
<td>577</td>
</tr>
<tr>
<td>in sinh()</td>
<td>578</td>
</tr>
<tr>
<td>in sqrt()</td>
<td>582</td>
</tr>
<tr>
<td>in srand48()</td>
<td>584</td>
</tr>
<tr>
<td>in str()</td>
<td>587</td>
</tr>
<tr>
<td>in strftime()</td>
<td>591</td>
</tr>
<tr>
<td>in strlen()</td>
<td>596</td>
</tr>
<tr>
<td>in strerror()</td>
<td>600</td>
</tr>
<tr>
<td>in strftime()</td>
<td>601</td>
</tr>
<tr>
<td>in strftime()</td>
<td>605-606</td>
</tr>
<tr>
<td>in strptime()</td>
<td>614</td>
</tr>
<tr>
<td>in strtod()</td>
<td>620-621</td>
</tr>
<tr>
<td>in strtof()</td>
<td>623-624</td>
</tr>
<tr>
<td>in strtoiu()</td>
<td>625-626</td>
</tr>
<tr>
<td>in strxfrm()</td>
<td>627</td>
</tr>
<tr>
<td>in swab()</td>
<td>629</td>
</tr>
<tr>
<td>in sysconf()</td>
<td>634-635</td>
</tr>
<tr>
<td>in tan()</td>
<td>640</td>
</tr>
<tr>
<td>in tanh()</td>
<td>641</td>
</tr>
<tr>
<td>in tcflush() (FIPS)</td>
<td>646</td>
</tr>
<tr>
<td>in tcgetattr()</td>
<td>648</td>
</tr>
<tr>
<td>in tcgetpgrp() (FIPS)</td>
<td>649</td>
</tr>
<tr>
<td>in tcsendbreak() (FIPS)</td>
<td>651</td>
</tr>
<tr>
<td>in tcsgetattr() (FIPS)</td>
<td>653</td>
</tr>
<tr>
<td>in tcssetpgrp() (FIPS)</td>
<td>655</td>
</tr>
<tr>
<td>in tdelete()</td>
<td>656</td>
</tr>
<tr>
<td>in telldir()</td>
<td>657</td>
</tr>
<tr>
<td>in tmpnam()</td>
<td>658</td>
</tr>
<tr>
<td>in tfind()</td>
<td>659</td>
</tr>
<tr>
<td>in time()</td>
<td>662</td>
</tr>
<tr>
<td>in tmpfile()</td>
<td>663</td>
</tr>
<tr>
<td>in toascii()</td>
<td>665</td>
</tr>
<tr>
<td>in tcseach()</td>
<td>673</td>
</tr>
<tr>
<td>in ttymname()</td>
<td>677</td>
</tr>
<tr>
<td>in twalk()</td>
<td>679</td>
</tr>
<tr>
<td>in tzset()</td>
<td>681</td>
</tr>
<tr>
<td>in ulimit()</td>
<td>684</td>
</tr>
<tr>
<td>in unlink()</td>
<td>690</td>
</tr>
<tr>
<td>in unlink() (FIPS)</td>
<td>689</td>
</tr>
<tr>
<td>in utime() (FIPS)</td>
<td>693</td>
</tr>
<tr>
<td>in wcstod()</td>
<td>722-723</td>
</tr>
<tr>
<td>in wcstombs()</td>
<td>727</td>
</tr>
<tr>
<td>in write()</td>
<td>739, 741-742</td>
</tr>
<tr>
<td>in write() (FIPS)</td>
<td>739</td>
</tr>
</tbody>
</table>
## Index

<table>
<thead>
<tr>
<th>Function</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in y0()</td>
<td>744</td>
</tr>
<tr>
<td>in _tolower()</td>
<td>666</td>
</tr>
<tr>
<td>in _toupper()</td>
<td>668</td>
</tr>
<tr>
<td>EXDEV</td>
<td>30, 751</td>
</tr>
<tr>
<td>in link()</td>
<td>356</td>
</tr>
<tr>
<td>in rename()</td>
<td>490</td>
</tr>
<tr>
<td>exec</td>
<td>131</td>
</tr>
<tr>
<td>execution</td>
<td>692</td>
</tr>
<tr>
<td>exit()</td>
<td>136</td>
</tr>
<tr>
<td>EXIT_FAILURE</td>
<td>807</td>
</tr>
<tr>
<td>EXIT_SUCCESS</td>
<td>807</td>
</tr>
<tr>
<td>exp()</td>
<td>138</td>
</tr>
<tr>
<td>expm1()</td>
<td>139</td>
</tr>
<tr>
<td>expressions</td>
<td>470</td>
</tr>
<tr>
<td>regular</td>
<td>470</td>
</tr>
<tr>
<td>Expr_Nest_Max</td>
<td>634, 772</td>
</tr>
</tbody>
</table>

## Extension

<table>
<thead>
<tr>
<th>Function</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EL</td>
<td>11</td>
</tr>
<tr>
<td>EX</td>
<td>11</td>
</tr>
<tr>
<td>FIPS</td>
<td>11</td>
</tr>
<tr>
<td>OH</td>
<td>12</td>
</tr>
<tr>
<td>WP</td>
<td>12</td>
</tr>
<tr>
<td>F-LOCK</td>
<td>858</td>
</tr>
<tr>
<td>fabs()</td>
<td>140</td>
</tr>
<tr>
<td>fattach()</td>
<td>141</td>
</tr>
<tr>
<td>fchdir()</td>
<td>143</td>
</tr>
<tr>
<td>fchmod()</td>
<td>144</td>
</tr>
<tr>
<td>fchown()</td>
<td>145</td>
</tr>
<tr>
<td>fclose()</td>
<td>146</td>
</tr>
<tr>
<td>fcntl()</td>
<td>148</td>
</tr>
<tr>
<td>fcvt()</td>
<td>120, 152</td>
</tr>
<tr>
<td>fdetach()</td>
<td>154</td>
</tr>
<tr>
<td>fdopen()</td>
<td>155</td>
</tr>
<tr>
<td>FD_CLOEXEC</td>
<td>74, 132, 148, 421, 753</td>
</tr>
<tr>
<td>FD_CLR</td>
<td>834</td>
</tr>
<tr>
<td>FD_CLR()</td>
<td>153</td>
</tr>
<tr>
<td>FD_ISSET</td>
<td>834</td>
</tr>
<tr>
<td>FD_SET</td>
<td>834</td>
</tr>
<tr>
<td>fd_set()</td>
<td>834</td>
</tr>
<tr>
<td>FD_SETSIZE</td>
<td>834</td>
</tr>
<tr>
<td>FD_ZERO</td>
<td>834</td>
</tr>
<tr>
<td>feof()</td>
<td>157</td>
</tr>
<tr>
<td>ferror()</td>
<td>158</td>
</tr>
<tr>
<td>FFDLY</td>
<td>848</td>
</tr>
<tr>
<td>fflush()</td>
<td>159</td>
</tr>
<tr>
<td>FFn</td>
<td>848</td>
</tr>
<tr>
<td>ffs()</td>
<td>161</td>
</tr>
<tr>
<td>fgetc()</td>
<td>162</td>
</tr>
<tr>
<td>fgetpos()</td>
<td>164</td>
</tr>
<tr>
<td>fgets()</td>
<td>165</td>
</tr>
<tr>
<td>fgetwc()</td>
<td>166</td>
</tr>
<tr>
<td>fgetws()</td>
<td>168</td>
</tr>
<tr>
<td>FIFOTYPE</td>
<td>844</td>
</tr>
<tr>
<td>FILENAME_MAX</td>
<td>804</td>
</tr>
<tr>
<td>fileno()</td>
<td>169</td>
</tr>
<tr>
<td>FIPS</td>
<td>11</td>
</tr>
<tr>
<td>FIPS 151-2</td>
<td>10</td>
</tr>
<tr>
<td>FIPS alignment</td>
<td>11</td>
</tr>
<tr>
<td>in &lt;errno.h&gt;</td>
<td>751</td>
</tr>
<tr>
<td>in &lt;limits.h&gt;</td>
<td>770, 773</td>
</tr>
<tr>
<td>in &lt;signal.h&gt;</td>
<td>795</td>
</tr>
<tr>
<td>in &lt;unistd.h&gt;</td>
<td>856</td>
</tr>
<tr>
<td>in access()</td>
<td>43</td>
</tr>
<tr>
<td>in chdir()</td>
<td>82</td>
</tr>
<tr>
<td>in chmod()</td>
<td>84</td>
</tr>
<tr>
<td>in chown()</td>
<td>86</td>
</tr>
<tr>
<td>in exec()</td>
<td>132-133</td>
</tr>
<tr>
<td>in fopen()</td>
<td>179</td>
</tr>
<tr>
<td>in freopen()</td>
<td>198</td>
</tr>
<tr>
<td>in kill()</td>
<td>344</td>
</tr>
<tr>
<td>in link()</td>
<td>355</td>
</tr>
<tr>
<td>in mkdir()</td>
<td>388</td>
</tr>
<tr>
<td>in mkfifo()</td>
<td>390</td>
</tr>
<tr>
<td>in open()</td>
<td>421, 423</td>
</tr>
<tr>
<td>in opendir()</td>
<td>425</td>
</tr>
<tr>
<td>in pathconf()</td>
<td>430</td>
</tr>
<tr>
<td>in read()</td>
<td>460</td>
</tr>
<tr>
<td>in rename()</td>
<td>490</td>
</tr>
<tr>
<td>in rmdir()</td>
<td>496</td>
</tr>
<tr>
<td>in setgid()</td>
<td>516</td>
</tr>
<tr>
<td>in setuid()</td>
<td>535</td>
</tr>
<tr>
<td>in stat()</td>
<td>587</td>
</tr>
<tr>
<td>in tcflush()</td>
<td>646</td>
</tr>
<tr>
<td>in tcgetpgrp()</td>
<td>649</td>
</tr>
<tr>
<td>in tcsendbreak()</td>
<td>651</td>
</tr>
<tr>
<td>in tcsetattr()</td>
<td>653</td>
</tr>
<tr>
<td>in tcsetpgrp()</td>
<td>655</td>
</tr>
<tr>
<td>in unlink()</td>
<td>689</td>
</tr>
<tr>
<td>in utime()</td>
<td>693</td>
</tr>
<tr>
<td>in write()</td>
<td>739</td>
</tr>
<tr>
<td>floor()</td>
<td>170</td>
</tr>
<tr>
<td>FLT_constants</td>
<td>756</td>
</tr>
<tr>
<td>defined in &lt;float.h&gt;</td>
<td>756</td>
</tr>
<tr>
<td>FLT_DIG</td>
<td>775</td>
</tr>
<tr>
<td>FLT_MANT_DIG</td>
<td>756</td>
</tr>
<tr>
<td>FLT_MAX</td>
<td>775</td>
</tr>
<tr>
<td>FLT_RADIX</td>
<td>756</td>
</tr>
<tr>
<td>fmod()</td>
<td>172</td>
</tr>
<tr>
<td>fmtmsg()</td>
<td>173</td>
</tr>
<tr>
<td>fnmatch()</td>
<td>176</td>
</tr>
</tbody>
</table>
Index

FNM_constants  
in <fnmatch.h> ................................................. 759
fopen() .............................................................. 178
FOPEN_MAX ....................................................... 804
fork() .................................................................. 181
format of entries ................................................... 13
fpathconf() .......................................................... 183
fprintf() .............................................................. 184
fputc() ................................................................ 190
fputs() ................................................................ 192
fputwc() .............................................................. 193
fputws() ................................................................ 195
fread() .................................................................. 196
free() ................................................................. 197
freopen() ................................................................ 198
frexp() .................................................................. 200
fscanf() .............................................................. 201
fseek() .................................................................. 207
fsetpos() .............................................................. 209
fstat() .................................................................. 210
fstatvfs() ............................................................. 212
fsync() .................................................................. 214
ftime() .................................................................. 215
ftell() .................................................................. 216
ftok() .................................................................. 217
ftruncate() ........................................................... 218
FTW .................................................................... 760
ftw() .................................................................. 220
FTW_constants in <ftw.h> .................................... 760
fwrite() ............................................................... 223
F_DUPFD ............................................................ 753
F_GETFD ........................................................... 753
F_GETFL ........................................................... 753
F_GETLK ........................................................... 753
F_OK ................................................................... 857
F_RDONLY .......................................................... 753
F_SETFD ........................................................... 753
F_SETFL ........................................................... 753
F_SETLKW .......................................................... 753
F_TEST ............................................................... 858
F_TIGHT ............................................................ 858
F_UNLOCK .......................................................... 858
gamma() ............................................................. 224
gcv() ................................................................ 120, 225
generating random numbers .................................. 300
GETALL .............................................................. 827
getc() .................................................................. 226
getchar() ............................................................ 227
getcontext() .......................................................... 228
getcwd() .............................................................. 229
getdate() ............................................................. 230
getdtablesize() .................................................... 234
getegid() ............................................................. 235
getenv() .............................................................. 236
geteuid() ............................................................. 237
getgid() ............................................................ 238
getgrent() ........................................................... 123, 239
getgrgid() ........................................................... 240
getgrnam() ........................................................... 241
getgroups() ......................................................... 242
gethostid() .......................................................... 243
gettimeofday() ..................................................... 244
getlogin() ............................................................ 246
getmsg() ............................................................... 248
GETPID ............................................................... 827
getopt() ............................................................... 251
getpagesize() ...................................................... 254
getpass() ............................................................. 255
getpgid() ............................................................. 257
getpgrp() ............................................................ 258
GETVAL .............................................................. 827
getpid() ............................................................... 259
getpmem() ........................................................... 248, 260
getppid() ............................................................. 261
getpriority() ......................................................... 262
getpwent() ........................................................... 124, 264
getpwnam() ........................................................ 265
getpwuid() .......................................................... 267
getreuid() ............................................................ 269
getresrc() ............................................................ 271
getresuid() .......................................................... 272
gets() .................................................................. 273
getsid() ............................................................... 274
getsubopt() ......................................................... 274
gmtime() ............................................................. 275
gettimeofday() ..................................................... 275
getuid() ............................................................... 276
getutsent() .......................................................... 277
getutxid() ........................................................... 125, 277
getutxline() ........................................................ 125
GETVAL .............................................................. 827
getw() .................................................................. 278
getwchar() ........................................................... 279
getwd() ............................................................... 280
GETALL .............................................................. 278
GLOB_constants  
defined in <glob.h> ................................................ 762
error returns of glob() .......................................... 283
Index

used in glob() ............................................................. 282

gmtim() ................................................................. 286
gnanpt() ................................................................. 287

granularity of clock ............................................... 216
hcreate() .............................................................. 288

hdestroy() ............................................................. 289
headers .................................................................. 745

hsearch() ............................................................... 290

HUGE_VAL ............................................................. 780

in ceil() ............................................................... 77

in clog() .............................................................. 102

in exp() ............................................................... 138

in floor() ............................................................. 170

in hypt() ............................................................... 293

in ldexp() ........................................................... 351

in lgamma() .......................................................... 354

in log() ................................................................. 365

in log10() ............................................................ 366

in pow() .............................................................. 440

in sinh() .............................................................. 578

in strtod() .......................................................... 620

in tan() ............................................................... 640

in wctod() .......................................................... 722

in y0() ................................................................. 744

HUPCL ................................................................. 848

hypot() ............................................................... 293

ICANON ................................................................ 849

iconv() ............................................................... 294

iconv_close() .................................................... 296

iconv_open() .................................................... 297

ICRNL ................................................................. 847

id_t ................................................................. 838

IEXTEN ............................................................... 849

IGNBRK .............................................................. 847

IGNCR ............................................................... 847

IGNPAR .............................................................. 847

flogb() ............................................................... 298

implementation-dependent .................................. 9
index() .............................................................. 299

Inf

in acosl() ........................................................... 45

in asinl() .......................................................... 51

in ceil() ............................................................. 77

in floor() ........................................................... 170

in fmod() .......................................................... 172

initstate() .......................................................... 300

INLCR ............................................................... 847

ino_t ................................................................. 838

INPCK ............................................................... 847

insque() ............................................................. 302

interfaces ................................................................ 15

file system .......................................................... 16

implementation .................................................. 15

system .............................................................. 39, 745

use ..................................................................... 15

interprocess communication .................................. 37

interval timers

changing timeout .................................................. 683

setting timeout ................................................... 683

INT_MAX ............................................................ 775

INT_MIN ............................................................ 776

invariant values .................................................... 776

ioctl() ............................................................... 304

iovec ............................................................... 840

IOV_MAX .......................................................... 635, 770

IPC ................................................................. 37

IPC_constants

defined in <sys/ipc.h> ......................................... 820

used in semctl() .................................................. 506

used in shmtct() .................................................. 540

isalnum() ............................................................ 315

isalpha() ............................................................. 316

iscanit() .............................................................. 317

isastream() .......................................................... 318

isatty() ............................................................... 319

iscntrl() .............................................................. 320

isdigit() .............................................................. 321

isgraph() ............................................................. 322

ISIG ................................................................. 849

islower() ............................................................. 323

isnan() ............................................................... 324

ISO C .................................................................. 16

isprint() ............................................................. 325

isprintc() ............................................................ 326

isspace() ............................................................. 327

ISIG .................................................................. 849

Issue 3

changes from .......................................................... 5

Issue 4

changes from .......................................................... 5

ISTRIP ............................................................... 847

isupper() .............................................................. 328

iswalnum() ........................................................... 329

iswalpha() ............................................................ 330

iswcntrl() ............................................................ 331

iswctype() ............................................................ 332

iswdir() .............................................................. 333

iswgraph() ............................................................ 334

iswlower() ............................................................ 335

iswprintf() ........................................................... 336

iswpcnt() ............................................................. 337

iswspace() ........................................................... 338
Index

isupper() .......................................................... 339
iswxdigit() .......................................................... 340
isxdigit() .......................................................... 341
itimerval .......................................................... 834
ITIMER_PROF .................................................. 834
ITIMER_REAL .................................................. 834
ITIMER_VIRTUAL ............................................ 834
IUCV ............................................................... 847
IXANY ............................................................ 847
IXOFF ............................................................. 847
IXON .............................................................. 847
j0() ................................................................ 342
jrand48() .......................................................... 116, 343
key_t ................................................................. 838
kill() ................................................................ 344
killpg() ............................................................. 346
l64a() ................................................................. 40, 347
labs() ................................................................. 348
LANG
  in <nl_types.h> .................................................. 785
  in catopen() ..................................................... 74
  lchown() .......................................................... 349
  lcong48() .......................................................... 116, 350
LC_ALL
  in localeconv() .................................................. 359
  in nl_langinfo() .................................................. 419
  in setlocale() ................................................... 523
LC_COLLATE
  in glob() .......................................................... 282-283
  in regexp .......................................................... 482
  in setlocale() ................................................... 523
  in strftime() ..................................................... 596
  in strxfrm() ..................................................... 627
  in wcsbcast() ................................................... 711
  in wcsxfrm() .................................................... 732
LC_CTYPE .................................................... 483, 734
  in iswctype() .................................................... 332
  in mblen() ....................................................... 379
  in mbstowcs() ................................................... 380
  in mbtowc() ...................................................... 381
  in setlocale() ................................................... 523
  in tolower() ...................................................... 667
  in toupper() ...................................................... 669
  in towlower() .................................................... 670
  in towupper() .................................................... 671
  in wctomb() ...................................................... 727
  in wctombs() ..................................................... 733
  in wcyan() ....................................................... 734
LC_MESSAGES
  in catopen() ..................................................... 74
  in setlocale() ................................................... 523-524
  in strerror() .................................................... 600
LC_MONETARY
  in localeconv() .................................................. 359
  in setlocale() ................................................... 523
  in strftime() .................................................... 602
LC_NUMERIC
  in fprintf() ...................................................... 184
  in fscanf() ....................................................... 201
  in localeconv() .................................................. 359
  in setlocale() ................................................... 523
  in strftime() .................................................... 602
  in strtok() ....................................................... 620
  in wcstod() ...................................................... 722
LC_TIME
  in setlocale() ................................................... 523
LDBL_ constants
  defined in <float.h> ............................................ 756
ldexp() .............................................................. 351
ldiv() ............................................................... 352
lfind() ............................................................... 353
lgamma() ............................................................ 354
limit
  numerical .......................................................... 775
  line control ....................................................... 849
LINE_MAX ....................................................... 634, 773
link() ............................................................... 355
LINK_MAX ......................................................... 429, 771
LNKTYPE ......................................................... 844
loc1 ................................................................. 357
local modes ....................................................... 849
localeconv() ...................................................... 358
localtime() ....................................................... 361
lockf() ............................................................... 362
locs ................................................................. 364
log() ................................................................. 365
log10() .............................................................. 366
log1p() .............................................................. 367
logb() ............................................................... 368
LOG_constants in syslog() ....................................... 95
longjmp() .......................................................... 370
LONG_BIT ......................................................... 775
LONG_MAX ......................................................... 775
LONG_MIN ......................................................... 776
lrand48() .......................................................... 116, 372
lseek() ............................................................... 373
lseek() ............................................................... 375
L_tmpnam .......................................................... 804
L_cuserid .......................................................... 804
MAGIC .............................................................. 747
makecontext() ..................................................... 377
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>malloc()</td>
<td>378</td>
</tr>
<tr>
<td>manual()</td>
<td></td>
</tr>
<tr>
<td>format</td>
<td>13</td>
</tr>
<tr>
<td>MAP_FIXED</td>
<td>822</td>
</tr>
<tr>
<td>MAP_PRIVATE</td>
<td>822</td>
</tr>
<tr>
<td>MAP_SHARED</td>
<td>822</td>
</tr>
<tr>
<td>MAXFLOAT</td>
<td>780</td>
</tr>
<tr>
<td>MAX_CANON</td>
<td>429,771</td>
</tr>
<tr>
<td>MAX_INPUT</td>
<td>429,771</td>
</tr>
<tr>
<td>may</td>
<td>9</td>
</tr>
<tr>
<td>mblen()</td>
<td>379</td>
</tr>
<tr>
<td>mbstowcs()</td>
<td>380</td>
</tr>
<tr>
<td>mbtowc()</td>
<td>381</td>
</tr>
<tr>
<td>MB_CUR_MAX</td>
<td>807</td>
</tr>
<tr>
<td>MB_LEN_MAX</td>
<td>775</td>
</tr>
<tr>
<td>mcontext_t_MAX</td>
<td>853</td>
</tr>
<tr>
<td>memccpy()</td>
<td>382</td>
</tr>
<tr>
<td>memchr()</td>
<td>383</td>
</tr>
<tr>
<td>memcmp()</td>
<td>384</td>
</tr>
<tr>
<td>memcpy()</td>
<td>385</td>
</tr>
<tr>
<td>memmove()</td>
<td>386</td>
</tr>
<tr>
<td>memset()</td>
<td>387</td>
</tr>
<tr>
<td>message catalog</td>
<td></td>
</tr>
<tr>
<td>in exec</td>
<td>132,134</td>
</tr>
<tr>
<td>in exit</td>
<td>136</td>
</tr>
<tr>
<td>minimum values</td>
<td>773</td>
</tr>
<tr>
<td>MINSIGSTKSZ</td>
<td>796</td>
</tr>
<tr>
<td>mkdir()</td>
<td>388</td>
</tr>
<tr>
<td>mkfifo()</td>
<td>390</td>
</tr>
<tr>
<td>mkmod()</td>
<td>392</td>
</tr>
<tr>
<td>mkstemp()</td>
<td>394</td>
</tr>
<tr>
<td>mktemp()</td>
<td>395</td>
</tr>
<tr>
<td>mktime()</td>
<td>396</td>
</tr>
<tr>
<td>mmap()</td>
<td>398</td>
</tr>
<tr>
<td>MM_macros</td>
<td>757</td>
</tr>
<tr>
<td>mode_t</td>
<td>838</td>
</tr>
<tr>
<td>modf()</td>
<td>401</td>
</tr>
<tr>
<td>MON_n</td>
<td>765</td>
</tr>
<tr>
<td>MORECTL</td>
<td>816</td>
</tr>
<tr>
<td>MOREDATA</td>
<td>816</td>
</tr>
<tr>
<td>mprotect()</td>
<td>402</td>
</tr>
<tr>
<td>mrand48()</td>
<td>116,403</td>
</tr>
<tr>
<td>msgctl()</td>
<td>404</td>
</tr>
<tr>
<td>msgget()</td>
<td>406</td>
</tr>
<tr>
<td>msgsnd()</td>
<td>408</td>
</tr>
<tr>
<td>MS_ANY</td>
<td>410</td>
</tr>
<tr>
<td>MSG_BAND</td>
<td>816</td>
</tr>
<tr>
<td>MSG_HIPRI</td>
<td>816</td>
</tr>
<tr>
<td>msgsync()</td>
<td>412</td>
</tr>
<tr>
<td>MS_ASYNC</td>
<td>822</td>
</tr>
<tr>
<td>MS_INVALIDATE</td>
<td>822</td>
</tr>
<tr>
<td>MS_SYNC</td>
<td>822</td>
</tr>
<tr>
<td>munmap()</td>
<td>414</td>
</tr>
<tr>
<td>M_ constants</td>
<td></td>
</tr>
<tr>
<td>defined in &lt;math.h&gt;</td>
<td>780</td>
</tr>
<tr>
<td>name space</td>
<td></td>
</tr>
<tr>
<td>X/Open</td>
<td>17</td>
</tr>
<tr>
<td>NAME_MAX</td>
<td>771</td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>in acos()</td>
<td>45</td>
</tr>
<tr>
<td>in asin()</td>
<td>51</td>
</tr>
<tr>
<td>in atan()</td>
<td>54</td>
</tr>
<tr>
<td>in atan2()</td>
<td>55</td>
</tr>
<tr>
<td>in ceil()</td>
<td>77</td>
</tr>
<tr>
<td>in cos()</td>
<td>101</td>
</tr>
<tr>
<td>in cosh()</td>
<td>102</td>
</tr>
<tr>
<td>in erf()</td>
<td>129</td>
</tr>
<tr>
<td>in exp()</td>
<td>138</td>
</tr>
<tr>
<td>in fabs()</td>
<td>140</td>
</tr>
<tr>
<td>in floor()</td>
<td>170</td>
</tr>
<tr>
<td>in fmod()</td>
<td>172</td>
</tr>
<tr>
<td>in printf()</td>
<td>186-187</td>
</tr>
<tr>
<td>in frexp()</td>
<td>200</td>
</tr>
<tr>
<td>in fscanf()</td>
<td>205</td>
</tr>
<tr>
<td>in hypot()</td>
<td>293</td>
</tr>
<tr>
<td>in isnan()</td>
<td>324</td>
</tr>
<tr>
<td>in j0()</td>
<td>342</td>
</tr>
<tr>
<td>in ldexp()</td>
<td>351</td>
</tr>
<tr>
<td>in lgamma()</td>
<td>354</td>
</tr>
<tr>
<td>in log()</td>
<td>365</td>
</tr>
<tr>
<td>in log10()</td>
<td>366</td>
</tr>
<tr>
<td>in modf()</td>
<td>401</td>
</tr>
<tr>
<td>in pow()</td>
<td>440</td>
</tr>
<tr>
<td>in sin()</td>
<td>577</td>
</tr>
<tr>
<td>in sinh()</td>
<td>578</td>
</tr>
<tr>
<td>in sqrt()</td>
<td>582</td>
</tr>
<tr>
<td>in tan()</td>
<td>640</td>
</tr>
<tr>
<td>in tanh()</td>
<td>641</td>
</tr>
<tr>
<td>in y0()</td>
<td>744</td>
</tr>
<tr>
<td>NCCS</td>
<td>846</td>
</tr>
<tr>
<td>NDEBUG</td>
<td>746</td>
</tr>
<tr>
<td>nextafter()</td>
<td>415</td>
</tr>
<tr>
<td>nftw()</td>
<td>416</td>
</tr>
<tr>
<td>NGROUPS_MAX</td>
<td>634,773</td>
</tr>
<tr>
<td>nice()</td>
<td>418</td>
</tr>
<tr>
<td>NLDLY</td>
<td>847</td>
</tr>
<tr>
<td>nlink_t</td>
<td>838</td>
</tr>
<tr>
<td>NLn</td>
<td>847</td>
</tr>
<tr>
<td>NLS_PATH</td>
<td></td>
</tr>
<tr>
<td>in catopen()</td>
<td>74-75</td>
</tr>
</tbody>
</table>
System Interfaces and Headers Issue 4, Version 2

Index

NL_ARGMAX.........................776
nl_langinfo()..................419
NL_LANGMAX.....................776
NL_MSGMAX......................776
NL_NMAX..........................776
NL_SETD.........................785
NL_SETMAX......................776
NL_TEXTMAX......................776
NOEXPR.........................766
NOREP..........................766
nrand48()......................116, 420
NULL...........................803-804, 807, 810, 851, 857
numerical limits...............775
NZERO............................776
OCRNL.........................847
off_t............................838
OFILL...........................847
OH.................................12
in chmod()......................84
in chown()......................86
in closedir()..................94
in creat().......................103
in fcntl().......................148
in fgetwc().....................166
in fgetws().....................168
in fork().......................181
in fputwc()...................193
in fputws()...................195
in fstat().......................210
in getegid()...................235
in geteuid()...................237
in getgid()....................238
in getgrgid()..................240
in getgrnam()..................241
in getgroups().................242
in getgrent()..................258
in getgid()....................259
in getppid()..................261
in getpwnam()...............265
in getpwuid()...............267
in getuid()...................276
in getwc().....................279
in kill().......................344
in lseek().....................375
in mkdir()....................388
in mkfifo()....................390
in open()......................421
in opendir()..................463
in readdir()...................463
in regcomp()...................474
in rewinddir()...............493
in seekdir()..................502
in setgid()...................516
in setpgid()..................526
in setsid()...................533
in setuid()...................535
in stat().......................587
in tcgetpgrp().................649
in tcsgetpgrp()..............655
in umask().....................685
in ungetwc()...............688
in utime().....................693
in wait()......................700
in waitpid()...............707
OLCUC..........................847
ONLCL..........................847
ONLRET..........................847
ONOCR..........................847
open().........................421
opendir().....................425
openlog().....................95, 427
OPEN_MAX......................634, 770
OPOST..........................428
O_C_constants
  defined in <fcntl.h>........753
  used in open()..............421
PAGESIZE.....................635, 770
PAGE_SIZE....................770
PAREN.........................848
PARMRK.......................847
PARODD.......................848
PASS_MAX.....................634, 770
PATH............................99
  in confstr().................131
  in exec......................131
  pathconf()..................429
  pathname variable values..771
PATH_MAX......................429, 771
pause().......................432
pclose().......................433
perror().......................434
persistent connection (I_PLINK)........312
pid_t.........................838
pipe()..........................435
PIPE_BUF.....................429, 771
PM_STR.......................765
poll()..........................436
pollfd.........................786
popen() .......................................................... 438
portability .......................................................... 11
pow() .................................................................. 440
printf() .......................................................... 184, 442
PRIO_constants
   defined in <sys/resource.h> .................. 825
process
   descriptor table size ............................... 234
   returning and setting scheduling priority .... 262
   returning resource utilisation for ......... 271
   setting real and effective user ID's ......... 531
   suspending execution .............................. 692
process group
   returning and setting scheduling priority .... 262
PROT_READ constants
   in <sys/mman.h> .................................. 822
ptsname() .................................................... 443
putc() .......................................................... 444
putchar() ....................................................... 445
putenv() ........................................................ 446
putmsg() ......................................................... 447
putpmsg() ....................................................... 447
puts() ........................................................... 449
pututxline() .............................................. 125, 450
putw() ........................................................... 451
putwc() ........................................................ 452
putwchar() ....................................................... 453
P_ALL ......................................................... 842
P_GID .......................................................... 842
P_PID .......................................................... 842
P_tmpdir ....................................................... 804
qsort() ............................................................. 454
queue
   inserting element in .................................. 302
   removing element from ............................ 302
RADIXCHAR .................................................. 766
raise() .......................................................... 455
rand() ........................................................... 456
random numbers
   generating .............................................. 300
random() ................................................... 300, 458
RAND_MAX .................................................. 807
read() ........................................................... 459
readdir() ....................................................... 463
readdir() ....................................................... 463
readlink() ....................................................... 465
readv() .......................................................... 466
realloc() ....................................................... 467
realpath() ....................................................... 469
regcmp() ....................................................... 472
regcomp() ....................................................... 474
regexp ......................................................... 472, 479

Index

REGTYPE ...................................................... 844
regular expression
   simple ..................................................... 470
   regular expressions ............................... 470
REG CONSTANTS
   defined in <regex.h> ............................ 788
   error return values of regcomp() ........... 476
   used in regcomp() .................................. 474-475
remainder() ................................................... 486
remove() ....................................................... 487
remque function ......................................... 302
remq() ........................................................... 488
rename() ....................................................... 489
resource utilisation
   returning information on ...................... 271
rewind() ....................................................... 492
rewinddir() ................................................... 493
re_comp() ..................................................... 470
RE_DUP_MAX .................................................. 634, 773
re_exec() ....................................................... 470
rindex() ......................................................... 494
rint() ........................................................... 495
rlimit .......................................................... 825
rmkdir() ......................................................... 496
run-time values
   increasable .............................................. 772
   invariant .................................................. 770
rusage ......................................................... 825
R_OK .......................................................... 857
SA_constants
   declared in <signal.h> ...................... 796
sbrk() .......................................................... 64, 498
scalb() .......................................................... 499
scanf() .......................................................... 201, 500
SCHAR_MAX ................................................... 775
SCHAR_MIN ................................................... 776
scheduling priority
   returning and setting ......................... 262
SC_AEXIT_MAX ............................................. 635
seek48() ....................................................... 116, 501
seekdir() ....................................................... 502
SEEK_CUR ...................................................... 804, 857
SEEK_END ...................................................... 804, 857
SEEK_SET ...................................................... 804, 857
select() ........................................................ 503
semctl() ....................................................... 506
semget() ....................................................... 509
semop() ........................................................ 511
SETALL ........................................................ 827
setbuf() ....................................................... 514
setcontext() .............................................................. 228, 515
setgid() ............................................................... 516
setgrent() ............................................................ 123, 517
setitimer() ............................................................ 244, 518
setjmp() ............................................................... 520
setkey() ............................................................... 522
setlocale() ........................................................... 523
setlogmask() ....................................................... 95, 525
setpgid() ............................................................. 526
setpgrp() ............................................................. 528
setpriority() ....................................................... 262, 529
setpwnent() .......................................................... 124
setregid() ............................................................ 530
setreuid() ............................................................ 531
setrlimit() ............................................................ 269, 532
setsev() ............................................................... 533
setstate() ............................................................ 300, 534
setuid() ............................................................... 535
setutxent() ........................................................... 125, 536

SETVAL ............................................................... 827

setvbuf() ............................................................. 537
shm() ................................................................. 538
shmctl() .............................................................. 540
shmdt() ............................................................... 542
shmget() ............................................................. 543
SHMLBA ............................................................... 829

SHM_RDONLY ....................................................... 829

SHM_RND ............................................................. 829

should ................................................................. 9

SHRT_MAX .......................................................... 775

SHRT_MIN ........................................................... 776

SIGABRT ............................................................. 795

sighandler() ........................................................ 545

sigaddset() .......................................................... 553

SIGALRM .............................................................. 795

SIGALRM .............................................................. 795

signalstack() ....................................................... 554

SIGBUS ............................................................... 795

SIGCHLD ............................................................ 638, 795

SIGCONT ............................................................ 795

sigdelset() ........................................................... 556

sigemptyset() ........................................................ 557

sigfillset() ........................................................... 558

SIGFPE ............................................................... 795

sighold() ............................................................. 559

SIGHUP ............................................................... 795

SIGILL ................................................................. 795

siginfo_t ............................................................... 797

SIGINT ............................................................... 638, 795

siginterrupt() ....................................................... 560

sigismember() ....................................................... 561

SIGKILL ............................................................... 795

siglongjmp() ........................................................ 562

signal() ............................................................... 563

signgam ............................................................. 566

sigpause() ........................................................... 567

sigpending() ........................................................ 568

SIGPIPE ............................................................. 795

SIGPOLL .............................................................. 795

sigprocmask() ...................................................... 569

SIGPROF ............................................................. 795

SIGQUIT .............................................................. 638, 795

sigrelease() .......................................................... 571

SIGSEGV ............................................................. 795

sigsetjmp() .......................................................... 572

sigstack .............................................................. 796

sigstack() ............................................................ 574

SIGSTKSZ ............................................................. 796

SIGSTOP ............................................................ 795

sigsuspend() ........................................................ 576

SICSYS ............................................................... 795

SIGTERM ............................................................. 795

SIGTRAP .............................................................. 795

SIGTSTP .............................................................. 795

SIGTTIN .............................................................. 795

SIGTTOU ............................................................. 795

SIGURG ............................................................... 795

SIGUSR1 .............................................................. 795

SIGUSR2 .............................................................. 795

SIGVTALRM .......................................................... 795

SIGXCPU ............................................................. 795

SIGXFSZ ............................................................. 795

simple regular expression ........................................ 470

sin() ................................................................. 577

sinh() ................................................................. 578

size_t ................................................................. 838

sleep() ............................................................... 579

sprintf() ............................................................. 184, 581

sqrt() ................................................................. 582

strnd() ............................................................... 583

strnd() ............................................................... 116, 584

strncpy() ............................................................. 300, 585

sscanf() ............................................................... 201, 586

SSIZE_MAX .......................................................... 775

ssize_t ............................................................... 838

SS_DISABLE .......................................................... 796
Index

timeval .................................................................834
timezone .............................................................662
time_t .................................................................838
TMAGIC .............................................................844
TMAGLEN ............................................................844
tmpfile() .............................................................663
tmpnam() .............................................................664
TMP_MAX ............................................................776, 804
toascii() ..............................................................665
toexec() ..............................................................844
tolower() .............................................................667
TOREAD .............................................................844
TOSTOP ...............................................................849
toupper() .............................................................669
towlower() ...........................................................670
TOWRITE .............................................................844
towupper() ...........................................................671
truncate() .............................................................218, 672
tsearch() ..............................................................673
TSGID .................................................................844
TSUID .................................................................844
TSTX .................................................................844
ttyname() ............................................................677
tysslot() ..............................................................678
TUEXEC ..............................................................844
TUREAD ..............................................................844
TUWRITE .............................................................844
TVERSLEN ..........................................................844
twars() ..............................................................679
tzname ..............................................................680
TZNAME_MAX ......................................................634, 770
tzset() ..............................................................681
T_FMT .................................................................765
T_FMT_AMPM .........................................................765
ualarm() .............................................................683
UCHAR_MAX .......................................................775
ucontext_t ..........................................................853
uid_t .................................................................838
UINT_MAX ..........................................................775
ulimit() ..............................................................684
ULONG_MAX ..........................................................775
UL_GETFSIZE .......................................................854
UL_SETFSIZE .......................................................854
umask() .............................................................685
uname() .............................................................686
undefined ...........................................................9
ungetc() .............................................................687
ungetwc() ...........................................................688
unlink() .............................................................689
unlockpt() ...........................................................691
...

user

returning and setting scheduling priority ...........................................262

user ID

real and effective ........................................531
setting real and effective ...............................................531

USHRT_MAX .........................................................775

usleep() ...........................................................692
utime() ...............................................................693
utimes() ..............................................................695
utmpx .................................................................864

UX .................................................................12, 17-21, 23, 25-30, 35, 38

in <cpio.h> ..........................................................747
in <errno.h> .........................................................750-751
in <fmtmsg.h> ......................................................757
in <ftw.h> ...........................................................760
in <grp.h> ...........................................................763
in <libgen.h> ........................................................768
in <limits.h> .........................................................770, 774
in <math.h> ..........................................................781
in <ndbm.h> ..........................................................784
in <poll.h> ...........................................................786
in <pwd.h> ...........................................................787
in <re_comp.h> .....................................................790
in <search.h> ........................................................792
in <setjmp.h> ........................................................793
in <signal.h> ........................................................795-796, 799
in <stdlib.h> .......................................................807-808
in <string.h> .......................................................810
in <strings.h> .......................................................812
in <stropts.h> ......................................................813
in <sys/ipc.h> ......................................................820
in <sys/mman.h> ...................................................822
in <sys/resource.h> ..............................................825
in <sys/stat.h> .....................................................830-831
in <sys/statvfs.h> ................................................833
in <sys/time.h> .....................................................834
in <sys/timeb.h> ...................................................836
in <sys/types.h> ....................................................838
in <sys/types.h> ....................................................838
in <sys/uio.h> .......................................................840
in <sys/wait.h> .....................................................842
in <syslog.h> ........................................................818
in <tar.h> ...........................................................844
in <termios.h> ......................................................849
in <time.h> ..........................................................851
in <ucontext.h> .....................................................853
in <unistd.h> .......................................................856-860
in <utmpx.h> .......................................................864
in a64l() .............................................................40
in access() .........................................................43
in acosh() ...........................................................46
Index

in alarm() ..............................................................48
in asinh() ..............................................................52
in atanh() ..............................................................56
in atexit() ..............................................................57
in basename() ..........................................................61
in bcmp() ..............................................................62
in bcopy() ..............................................................63
in brk() .................................................................64
in bsd_signal() ......................................................66
in bzero() ...............................................................70
in catgets() ...........................................................73
in catopen() ...........................................................74
in cbt() ..................................................................76
in cfsetispeed() .....................................................80
in cfsetspeed() ......................................................81
in chmod() ................................................................82
in chdir() ................................................................88
in chroot() ..............................................................90
in close() ..............................................................92-93
in closelog() ..........................................................95
in dbm_clearerr() ..................................................110
in dirname() ..........................................................114
in cvt() ..................................................................120
in endgrent() .........................................................123
in endpwent() .........................................................124
in endutxent() .......................................................125
in exec() ..............................................................132-134
in exit() ..............................................................136-137
in expm1() .............................................................139
in fattach() ...........................................................141
in fchdir() .............................................................143
in fchmod() .............................................................144
in fchown() ...........................................................145
in fcvt() ...............................................................152
in fdetach() ...........................................................154
in FD_CLR() ..........................................................153
in ffs() .................................................................161
in fgetct() .............................................................162
in fgetwc() ...........................................................166
in fmtmsg() ...........................................................173
in fopen() ............................................................179
in fork() ..............................................................181
in fprintf() ...........................................................188
in fputc() ..............................................................190
in fputwc() ...........................................................193
in free() ..............................................................197
in freopen() .........................................................198-199
in fseek() .............................................................207
in fstat() .............................................................210
in fstatvfs() ..........................................................212
in ftruncate() ........................................................218
in ftruncate() ........................................................218
in ftime() .............................................................216
in ftok() ...............................................................217
in ftw() ...............................................................220-221
in gcvt() ..............................................................225
in getcontext() ......................................................228
in getdate() ...........................................................230
in getdtablesize() ..................................................234
in getgrent() ........................................................239
in gethostid() ........................................................243
in getitimer() ........................................................244
in getmsg() ...........................................................248
in getpagesize() ....................................................254
in getpgid() ...........................................................257
in getpmem() ........................................................260
in getpriority() ......................................................262
in getpwent() ........................................................264
in getrlimit() ........................................................269
in getrusage() .......................................................271
in getsid() ............................................................273
in getsubopt() .......................................................274
in gettimeofday() ...................................................275
in getutxent() .......................................................277
in getwd() .............................................................281
in granpt() ...........................................................287
in ilogb() ...............................................................298
in index() .............................................................299
in initstate() .........................................................300
in insue() ............................................................302
in ioctl() .............................................................304
in isastream() .......................................................318
in killpg() ............................................................346
in l64a() ...............................................................347
in lchown() ...........................................................349
in link() .............................................................355-356
in lock() ..............................................................362
in log1p() ............................................................367
in logb() .............................................................368
in jpeg() ..............................................................370
in lstat() .............................................................376
in makecontext() ...................................................377
in mkdir() ...........................................................388
in mkfifo() ...........................................................390
in mknode() ..........................................................392
in mkstemp() .........................................................394
in mktemp() ..........................................................395
in mmax() .............................................................398
in mprotect() .......................................................402
in msync() ...........................................................412
in munmap() ........................................................414
in nextafter() .......................................................415
Index

in nftw( ) ..............................................................416
in nice( ) ............................................................418
in open( ) ..........................................................422-423
in opendir( ) ......................................................425
in openlog( ) ......................................................427
in pathconf( ) ....................................................430
in pipe( ) ..........................................................435
in poll( ) ...........................................................436
in ptsname( ) .....................................................443
in putmsg( ) ........................................................447
in putuxline( ) ....................................................450
in random( ) ......................................................458
in read( ) ..........................................................459-461
in readdir( ) ......................................................463
in readlink( ) .....................................................465
in readdir( ) ......................................................466
in realpath( ) ....................................................469
in regcomp( ) .....................................................472
in regex( ) ........................................................479
in remainder( ) ..................................................486
in remque( ) ......................................................488
in rename( ) ......................................................489-490
in re_comp( ) .....................................................470
in rindex( ) .......................................................494
in rint( ) ..........................................................495
in rmdir( ) ......................................................496-497
in sbrk( ) ..........................................................498
in scalar( ) .......................................................499
in seekdir( ) ......................................................502
in select( ) .......................................................503
in setcontext( ) ................................................515
in setgrent( ) ....................................................517
in setitimer( ) ....................................................518
in setlogmask( ) ................................................525
in setpgid( ) .....................................................528
in setpriority( ) ................................................529
in setregid( ) .....................................................530
in setreuid( ) ....................................................531
in setrlimit( ) ....................................................532
in setstate( ) ....................................................534
in setutxent( ) ..................................................536
in shmctl( ) .......................................................540
in sigaction( ) ...................................................545, 547-549, 551
in sigaltstack( ) ................................................554
in siginterrupt( ) ..............................................560
in signal( ) ......................................................563-564
in sigstack( ) .....................................................574
in sleep( ) .........................................................579
in srandom( ) ....................................................585
in stat( ) ..........................................................587
in statvfs( ) ......................................................589
in strcasecmp( ) ..................................................592
in strdup( ) ......................................................599
instrncmp( ) .....................................................609
in swapcontext( ) .............................................630
in symlink( ) .....................................................631
in sync( ) ........................................................633
in syslog( ) .......................................................635
in tcgetpgrp( ) ................................................650
in telldir( ) ......................................................657
in truncate( ) ...................................................]672
in ttyslot( ) ......................................................678
in ualarm( ) .....................................................683
in ulimit( ) ......................................................684
in unlink( ) .....................................................689-690
in unlockpt( ) ..................................................691
in usleep( ) ......................................................692
in utime( ) .......................................................693
in utimes( ) ......................................................695
in valloc( ) .......................................................696
in vfork( ) ........................................................697
in wait( ) .......................................................700-701
in wait3( ) .......................................................704
in waitid( ) .....................................................705
in write( ) .....................................................739-742
in _longjmp( ) ..................................................369
in _setjmp( ) ....................................................519
valloc( ) ..........................................................696
VEOF ..........................................................846
VEOL ..........................................................846
VERASE .........................................................846
vfork( ) ..........................................................697
vfprintf( ) ......................................................699
VINTR ..........................................................846
VKILL ...........................................................846
VQUIT ...........................................................846
VSTART .........................................................846
VSTOP ...........................................................846
VSUSP ..........................................................847
VTDL ...........................................................847
VTn ..............................................................847
wait( ) ...........................................................700
wait3( ) ..........................................................704
waitid( ) .......................................................705
waitpid( ) .......................................................707
warning ..........................................................12
UX ...............................................................12
WCONTINUED ................................................842
wcsccat( ) ......................................................708
wcschr( ) .......................................................709
wcsncmp( ) ......................................................710
Index

wcscoll() .......................................................... 711
wcscpy() .......................................................... 712
wcsncpy() .......................................................... 713
wcsftime() .......................................................... 714
wcslen() .......................................................... 715
wcsncat() .......................................................... 716
wcsncmp() .......................................................... 717
wcsncpy() .......................................................... 718
wcsrchr() .......................................................... 720
wcsnstr() .......................................................... 721
wcstol() .......................................................... 722
wcstok() .......................................................... 724
wcsxfrm() .......................................................... 727
wcwidth() .......................................................... 728
wcswcs() .......................................................... 730
wcswcs() .......................................................... 731
wcswparams() ....................................................... 732
wcwidth() .......................................................... 733
wcsxfrm() .......................................................... 734
wcwidth() .......................................................... 735
in <wchar.h> ....................................................... 867
in fgetwc() ......................................................... 166
in fgetws() ......................................................... 168
in fputwc() ......................................................... 193
in fputws() ......................................................... 195
in fseek() .......................................................... 207
in getwc() .......................................................... 279
in getwchar() ....................................................... 280
in isalnum() ......................................................... 329
in isalpha() ......................................................... 330
in isalnum() ......................................................... 331
in iswctype() ....................................................... 332
in iswctype() ....................................................... 333
in iswgraph() ....................................................... 334
in iswgraph() ....................................................... 335
in iswprint() ....................................................... 336
in iswprintf() ...................................................... 337
in iswspace() ....................................................... 338
in iswupper() ....................................................... 339
in iswxdigit() ....................................................... 340
in putwc() ......................................................... 452
in putwchar() ....................................................... 453
in tolower() ......................................................... 670
in toupper() ......................................................... 671
in ungetwc() ....................................................... 688
in wmemcpy() ......................................................... 708
in wmemcpy() ......................................................... 709
in wcscoll() ......................................................... 711
in wcsxfrm() ....................................................... 712
in wcsxfrm() ....................................................... 713
in wcswcs() ......................................................... 714
in wcslen() ......................................................... 715
in wcsncat() ....................................................... 716
in wcsncmp() ....................................................... 717
in wcsncpy() ....................................................... 718
in wcsxfrm() ....................................................... 719
in wcsxfrm() ....................................................... 720
in wcsxfrm() ....................................................... 721
in wcsxfrm() ....................................................... 722
in wcsxfrm() ....................................................... 723
in wcsxfrm() ....................................................... 724
in wcsxfrm() ....................................................... 725
in wcsxfrm() ....................................................... 726
in wcsxfrm() ....................................................... 727
in wcsxfrm() ....................................................... 728
in wcsxfrm() ....................................................... 729
in wcsxfrm() ....................................................... 730
in wcsxfrm() ....................................................... 731
in wcsxfrm() ....................................................... 732
in wcsxfrm() ....................................................... 733
in wcsxfrm() ....................................................... 734
in wcsxfrm() ....................................................... 735
WRDE_APPEND .................................................. 869
WRDE_BADCHAR .................................................. 869
WRDE_BADVAL .................................................. 869
WRDE_CMDSUB .................................................. 869
WRDE_DOFFS .................................................... 869
WRDE_NOCMD .................................................... 869
WRDE_NOSPACE ................................................... 869
WRDE_NOSYS ..................................................... 869
WRDE_REUSE ..................................................... 869
WRDE_SHOWER ................................................... 869
WRDE_SYNTAX .................................................... 869
WRDE_UNDEF .................................................... 869
write() .......................................................... 739
X/Open CAE Specification (1994)
Index

XCASE .................................................................849
X_OK .................................................................857
y0() ...............................................................744
YESEXPR .......................................................766
YESSTR .........................................................766