Technical Reference: Base Operating System and Extensions, Volume 2
Note
Before using this information and the product it supports, read the information in Appendix C, "Notices," on page 775.

Third Edition (September 2005)
This edition applies to AIX 5L Version 5.3 and to all subsequent releases of this product until otherwise indicated in new editions.

A reader's comment form is provided at the back of this publication. If the form has been removed, address comments to Information Development, Department 04XA-905-6C006, 11501 Burnet Road, Austin, Texas 78758-3493. To send comments electronically, use this commercial Internet address: aix6kpub@austin.ibm.com. Any information that you supply may be used without incurring any obligation to you.

© Copyright International Business Machines Corporation 1994, 2005. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
Contents

About This Book ........................................................................................................... xv
Highlighting .................................................................................................................... xv
Case-Sensitivity in AIX .................................................................................................. xv
ISO 9000 ....................................................................................................................... xv
32-Bit and 64-Bit Support for the Single UNIX Specification ...................................... xvi
Related Publications .................................................................................................... xvi

Chapter 1. Base Operating System (BOS) Runtime Services (Q-Z) .................................. 1
qsort Subroutine .......................................................................................................... 1
quotactl Subroutine ...................................................................................................... 2
raise Subroutine ......................................................................................................... 5
rand or srand Subroutine ......................................................................................... 6
rand_r Subroutine ...................................................................................................... 7
random, srand, initstate, or setstate Subroutine .................................................. 8
ra_attachrset Subroutine ....................................................................................... 10
ra_detachrset Subroutine ....................................................................................... 12
ra_exec Subroutine ................................................................................................. 14
ra_free_attachinfo Subroutine ........................................................................... 16
ra_get_attachinfo Subroutine ........................................................................... 18
ra_getrset Subroutine ............................................................................................ 21
ra_mmap or ra_mmapv Subroutine ...................................................................... 22
ra_shmget and ra_shmgetv Subroutines ................................................................ 26
read, readdir, readdirv, or pread Subroutine ....................................................... 29
readdir_r Subroutine .............................................................................................. 33
readdir_r Subroutine .............................................................................................. 35
read_real_time or time_base_to_time Subroutine ........................................... 36
realpath Subroutine ................................................................................................. 38
reboot Subroutine .................................................................................................... 39
re_comp or re_exec Subroutine ........................................................................... 40
regcmp or regex Subroutine .................................................................................. 41
regcomp Subroutine ............................................................................................... 44
regerror Subroutine ............................................................................................... 46
regex Subroutine ..................................................................................................... 47
regfree Subroutine .................................................................................................. 50
reltimerid Subroutine ............................................................................................ 51
remainder, remainderf, or remainderl Subroutine ........................................... 52
remove Subroutine .................................................................................................. 52
removeea Subroutine .............................................................................................. 53
remquo, remquof, or remquol Subroutine .......................................................... 54
rename Subroutine .................................................................................................. 55
reset_malloc_log Subroutine ............................................................................... 57
revoke Subroutine .................................................................................................... 58
rintf, rintl, or rint Subroutine ............................................................................... 59
rmproj Subroutine .................................................................................................. 60
rmprojdb Subroutine ............................................................................................. 61
round, roundf, or roundl Subroutine ................................................................... 62
rpmatch Subroutine ............................................................................................... 64
RSiAddSetHot Subroutine ..................................................................................... 65
RSiChangeFeed Subroutine ................................................................................... 68
RSiChangeHotFeed Subroutine ............................................................................. 69
RSiClose Subroutine ............................................................................................... 70

© Copyright IBM Corp. 1994, 2005
sem_post Subroutine .......................................................... 149
sem_timedwait Subroutine .................................................. 150
sem_trywait and sem_wait Subroutine ................................ 151
sem_unlink Subroutine ....................................................... 152
semct1 Subroutine ............................................................ 153
semget Subroutine ............................................................. 156
semop Subroutine .............................................................. 158
setacldb or endacldb Subroutine ....................................... 161
setauthdb or setauthdb_r Subroutine ................................ 162
setbuf, setvbuf, setbuffer, or setlinebuf Subroutine .......... 163
setcmmap Subroutine ......................................................... 165
seteal Subroutine ............................................................ 166
setgid, setegid, setregid, or setgidx Subroutine ............... 167
setgroups Subroutine ......................................................... 169
setjmp or longjmp Subroutine .......................................... 170
setlocale Subroutine ........................................................ 171
setpgvvalue or setpgvvalue64 Subroutine ...................... 174
setpcr Subroutine ............................................................ 175
setpenv Subroutine .......................................................... 178
setpgid or setpgrp Subroutine .......................................... 182
setpri Subroutine .............................................................. 183
setpwpdb or endpwpdb Subroutine .................................... 184
setroledb or endroledb Subroutine ................................ 185
setsid Subroutine ............................................................. 186
setuid, seteuid, setregid, or setgidx Subroutine ............... 187
setuserdb or enduserdb Subroutine .................................. 189
sgetl or spctl Subroutine .................................................. 190
shm_open Subroutine .......................................................... 191
shm_unlink Subroutine ....................................................... 193
shm Subroutine ................................................................. 194
shmct1 Subroutine ............................................................. 197
shmdt Subroutine ............................................................. 200
shmget Subroutine ............................................................. 201
sigaction, sigvec, or signal Subroutine ......................... 203
sigaltstack Subroutine ....................................................... 214
sigemptyset, sigfillset, sigaddset, sigdelset, or sigismember Subroutine ................................. 215
siginterrupt Subroutine ....................................................... 217
signbit Macro ................................................................. 218
sigpending Subroutine ......................................................... 218
sigprocmask, sigsetmask, or sigblock Subroutine ............... 219
sigqueue Subroutine ............................................................. 221
sigset, sighold, sigrealse, or sigignore Subroutine ........... 222
sigsetjmp or siglongjmp Subroutine ................................ 226
sigstack Subroutine ............................................................. 227
sigsuspend or sigpause Subroutine ................................ 228
sigthreadmask Subroutine ..................................................... 229
sigtimedwait and sigwaitinfo Subroutine ....................... 231
sigwait Subroutine ............................................................. 232
sin, sinf, or sinl Subroutine ........................................... 233
sinh, sinh, or sinh Subroutine ........................................... 234
sleep, nsleep or usleep Subroutine ................................ 236
sockatmark Subroutine ......................................................... 237
SpmiAddSetHot Subroutine ............................................... 238
SpmiCreateHotSet ............................................................... 241
SpmiCreateStatSet Subroutine ........................................... 242
SpmiDdsAddCx Subroutine ................................................... 243
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpmiDdsDelCx Subroutine</td>
<td>244</td>
</tr>
<tr>
<td>SpmiDdsInit Subroutine</td>
<td>245</td>
</tr>
<tr>
<td>SpmiDelSetHot Subroutine</td>
<td>247</td>
</tr>
<tr>
<td>SpmiDelSetStat Subroutine</td>
<td>248</td>
</tr>
<tr>
<td>SpmiExit Subroutine</td>
<td>250</td>
</tr>
<tr>
<td>SpmiFirstCx Subroutine</td>
<td>250</td>
</tr>
<tr>
<td>SpmiFirstHot Subroutine</td>
<td>251</td>
</tr>
<tr>
<td>SpmiFirstStat Subroutine</td>
<td>252</td>
</tr>
<tr>
<td>SpmiFirstVals Subroutine</td>
<td>254</td>
</tr>
<tr>
<td>SpmiFreeHotSet Subroutine</td>
<td>255</td>
</tr>
<tr>
<td>SpmiFreeStatSet Subroutine</td>
<td>256</td>
</tr>
<tr>
<td>SpmiGetCx Subroutine</td>
<td>257</td>
</tr>
<tr>
<td>SpmiGetHotSet Subroutine</td>
<td>258</td>
</tr>
<tr>
<td>SpmiGetStat Subroutine</td>
<td>259</td>
</tr>
<tr>
<td>SpmiGetStatSet Subroutine</td>
<td>261</td>
</tr>
<tr>
<td>SpmiGetValue Subroutine</td>
<td>262</td>
</tr>
<tr>
<td>SpmilInit Subroutine</td>
<td>263</td>
</tr>
<tr>
<td>SpmilInstantiate Subroutine</td>
<td>265</td>
</tr>
<tr>
<td>SpmiNextCx Subroutine</td>
<td>266</td>
</tr>
<tr>
<td>SpmiNextHot Subroutine</td>
<td>267</td>
</tr>
<tr>
<td>SpmiNextHotItem Subroutine</td>
<td>268</td>
</tr>
<tr>
<td>SpmiNextStat Subroutine</td>
<td>270</td>
</tr>
<tr>
<td>SpmiNextVals Subroutine</td>
<td>272</td>
</tr>
<tr>
<td>SpmiNextValue Subroutine</td>
<td>272</td>
</tr>
<tr>
<td>SpmiPathAddStatSet Subroutine</td>
<td>274</td>
</tr>
<tr>
<td>SpmiPathGetCx Subroutine</td>
<td>276</td>
</tr>
<tr>
<td>SpmiStatGetPath Subroutine</td>
<td>277</td>
</tr>
<tr>
<td>sqrt, sqrtf, or sqrtl Subroutine</td>
<td>278</td>
</tr>
<tr>
<td>src_err_msg Subroutine</td>
<td>280</td>
</tr>
<tr>
<td>src_err_msg_r Subroutine</td>
<td>281</td>
</tr>
<tr>
<td>srcrrqs Subroutine</td>
<td>281</td>
</tr>
<tr>
<td>srcrrqs_r Subroutine</td>
<td>281</td>
</tr>
<tr>
<td>srcsbuf Subroutine</td>
<td>284</td>
</tr>
<tr>
<td>srcsbuf_r Subroutine</td>
<td>287</td>
</tr>
<tr>
<td>srcsrpy Subroutine</td>
<td>290</td>
</tr>
<tr>
<td>srcsrqt Subroutine</td>
<td>293</td>
</tr>
<tr>
<td>srcsrqt_r Subroutine</td>
<td>296</td>
</tr>
<tr>
<td>srcstat Subroutine</td>
<td>299</td>
</tr>
<tr>
<td>srcstat_r Subroutine</td>
<td>302</td>
</tr>
<tr>
<td>srcstathdr Subroutine</td>
<td>304</td>
</tr>
<tr>
<td>srcstattxt Subroutine</td>
<td>305</td>
</tr>
<tr>
<td>srcstattxt_r Subroutine</td>
<td>305</td>
</tr>
<tr>
<td>srcstop Subroutine</td>
<td>306</td>
</tr>
<tr>
<td>srcstrt Subroutine</td>
<td>308</td>
</tr>
<tr>
<td>ssignal or gsignal Subroutine</td>
<td>311</td>
</tr>
<tr>
<td>statacl or fstatacl Subroutine</td>
<td>312</td>
</tr>
<tr>
<td>statea Subroutine</td>
<td>314</td>
</tr>
<tr>
<td>statfs, fstatfs, statfs64, or ustat Subroutine</td>
<td>315</td>
</tr>
<tr>
<td>statvfs, fstatvfs, statvfs64, or fstatvfs64 Subroutine</td>
<td>317</td>
</tr>
</tbody>
</table>
| statx, stat, lstat, fstat, fullstat, fullstat, stat64, lstat64, fstat64, stat64x, fstat64x, or lstat64x Subroutine | 318  
| strcat, strncat, strxfrm, strpcy, strmpy, or strdup Subroutine | 323  
| strmpy, strncmp, strcasecmp, strncasecmp, or strcoll Subroutine | 325  
| strerror Subroutine               | 327  |
| strftime Subroutine              | 328  |
| strftime Subroutine              | 330  |
strlen, strchr, strchr, strpbrk, strlen, strspn, strstr, strtok, or strsep Subroutine  333
strncollen Subroutine  336
strtof, strtod, or strtold Subroutine  337
strtoimax or str toumax Subroutine  339
strtok_r Subroutine  340
strtol, strtoul, strtoll, strtoull, or atoi Subroutine  341
strftime Subroutine  342
strtok Subroutine  343
swab Subroutine  345
swapoff Subroutine  346
swapon Subroutine  347
swapqry Subroutine  349
symlink Subroutine  350
sync Subroutine  352
syncvfs Subroutine  353
_sync_cache_range Subroutine  354
sysconf Subroutine  355
sysconfig Subroutine  358
SYS_CFGDD sysconfig Operation  360
SYS_CFGKMOD sysconfig Operation  361
SYS_GETLPAR_INFO sysconfig Operation  363
SYS_GETPARMS sysconfig Operation  364
SYS_KLOAD sysconfig Operation  364
SYS_KULOAD sysconfig Operation  367
SYS_QDVSW sysconfig Operation  368
SYS_QUERYLOAD sysconfig Operation  369
SYS_SETPARMS sysconfig Operation  370
SYS_SINGLELOAD sysconfig Operation  372
syslog, openlog, closelog, or setlogmask Subroutine  372
syslog_r, openlog_r, closelog_r, or setlogmask_r Subroutine  375
sysparm Subroutine  379
system Subroutine  381
tan, tanf, or tanl Subroutine  382
tanh, tanhf, or tanhl Subroutine  383
tcb Subroutine  384
tcdrain Subroutine  385
tcflow Subroutine  386
tcflush Subroutine  388
tcgetattr Subroutine  389
tcgetattr Subroutine  390
tcgetpgrp Subroutine  391
tcsendbreak Subroutine  392
tcgetattr Subroutine  394
tcsetattr Subroutine  395
tcsetpgrp Subroutine  396
termdef Subroutine  397
test_and_set Subroutine  398
tgamma, tgammaf, or tgammal Subroutine  397
timer_create Subroutine  398
timer_delete Subroutine  400
timer_getopterr, timer_gettime, and timer_settime Subroutine  400
times Subroutine  402
timezone Subroutine  404
thread_post Subroutine  405
thread_post_many Subroutine  406
thread_self Subroutine  407
thread_setsched Subroutine  407
thread_wait Subroutine  409
<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>wait, waitpid, wait3, or wait364 Subroutine</td>
<td>488</td>
</tr>
<tr>
<td>waitid Subroutine</td>
<td>491</td>
</tr>
<tr>
<td>wcsat, wcschr, wcscmp, wcscpy, or wcscspn Subroutine</td>
<td>493</td>
</tr>
<tr>
<td>wcscoll Subroutine</td>
<td>494</td>
</tr>
<tr>
<td>wcsftime Subroutine</td>
<td>496</td>
</tr>
<tr>
<td>wcscid Subroutine</td>
<td>497</td>
</tr>
<tr>
<td>wcslen Subroutine</td>
<td>497</td>
</tr>
<tr>
<td>wcsncat, wcscncmp, or wcscncpy Subroutine</td>
<td>498</td>
</tr>
<tr>
<td>wcspbrk Subroutine</td>
<td>499</td>
</tr>
<tr>
<td>wcsrchr Subroutine</td>
<td>500</td>
</tr>
<tr>
<td>wcstombs Subroutine</td>
<td>501</td>
</tr>
<tr>
<td>wcsspnp Subroutine</td>
<td>502</td>
</tr>
<tr>
<td>wcstr Subroutine</td>
<td>503</td>
</tr>
<tr>
<td>wcstol or wcstoll Subroutine</td>
<td>505</td>
</tr>
<tr>
<td>wcstok Subroutine</td>
<td>506</td>
</tr>
<tr>
<td>wcstol or wcstoll Subroutine</td>
<td>508</td>
</tr>
<tr>
<td>wcstombs Subroutine</td>
<td>510</td>
</tr>
<tr>
<td>wcstoul or wcstoull Subroutine</td>
<td>511</td>
</tr>
<tr>
<td>wcswcs Subroutine</td>
<td>513</td>
</tr>
<tr>
<td>wcswidth Subroutine</td>
<td>514</td>
</tr>
<tr>
<td>wcsxfrm Subroutine</td>
<td>515</td>
</tr>
<tr>
<td>wctob Subroutine</td>
<td>516</td>
</tr>
<tr>
<td>wcstrmb Subroutine</td>
<td>517</td>
</tr>
<tr>
<td>wctrans Subroutine</td>
<td>518</td>
</tr>
<tr>
<td>wctype or get_wctype Subroutine</td>
<td>518</td>
</tr>
<tr>
<td>wcwidth Subroutine</td>
<td>520</td>
</tr>
<tr>
<td>wlm_assign Subroutine</td>
<td>521</td>
</tr>
<tr>
<td>wlm_change_class Subroutine</td>
<td>523</td>
</tr>
<tr>
<td>wlm_check subroutine</td>
<td>525</td>
</tr>
<tr>
<td>wlm_classify Subroutine</td>
<td>526</td>
</tr>
<tr>
<td>wlm_class2key Subroutine</td>
<td>528</td>
</tr>
<tr>
<td>wlm_create_class Subroutine</td>
<td>528</td>
</tr>
<tr>
<td>wlm_delete_class Subroutine</td>
<td>530</td>
</tr>
<tr>
<td>wlm_endkey Subroutine</td>
<td>531</td>
</tr>
<tr>
<td>wlm_get_bio_stats subroutine</td>
<td>532</td>
</tr>
<tr>
<td>wlm_get_info Subroutine</td>
<td>535</td>
</tr>
<tr>
<td>wlm_get_proclinfo Subroutine</td>
<td>537</td>
</tr>
<tr>
<td>wlm_init_class_definition Subroutine</td>
<td>538</td>
</tr>
<tr>
<td>wlm_initialize Subroutine</td>
<td>539</td>
</tr>
<tr>
<td>wlm_initkey Subroutine</td>
<td>540</td>
</tr>
<tr>
<td>wlm_key2class Subroutine</td>
<td>541</td>
</tr>
<tr>
<td>wlm_load Subroutine</td>
<td>542</td>
</tr>
<tr>
<td>wlm_read_classes Subroutine</td>
<td>544</td>
</tr>
<tr>
<td>wlm_set Subroutine</td>
<td>545</td>
</tr>
<tr>
<td>wlm_set_tag Subroutine</td>
<td>547</td>
</tr>
<tr>
<td>wmemchr Subroutine</td>
<td>548</td>
</tr>
<tr>
<td>wmemcmp Subroutine</td>
<td>549</td>
</tr>
<tr>
<td>wmemcp Subroutine</td>
<td>550</td>
</tr>
<tr>
<td>wmemmove Subroutine</td>
<td>550</td>
</tr>
<tr>
<td>wmemset Subroutine</td>
<td>551</td>
</tr>
<tr>
<td>wordexp Subroutine</td>
<td>551</td>
</tr>
<tr>
<td>wordfree Subroutine</td>
<td>554</td>
</tr>
<tr>
<td>write, writex, writev, writexx or pwrite Subroutines</td>
<td>554</td>
</tr>
<tr>
<td>wstring Subroutine</td>
<td>560</td>
</tr>
<tr>
<td>wstrtod or watof Subroutine</td>
<td>563</td>
</tr>
</tbody>
</table>
**wstrtol, watol, or watoi Subroutine** .............................................. 564
wcrypt_key_setup, wcrypt_encrypt, wcrypt_decrypt, wcrypt_hash, wcrypt_malloc, wcrypt_free,
wcrypt_printb, wcrypt_mac, wcrypt_hmac, wcrypt_sign, wcrypt_verify, wcrypt_dh_keygen,
wcrypt_dh, wcrypt_btoa and wcrypt_randbuff Subroutine .............................................. 565
yield Subroutine ................................................................. 570

**Chapter 2. Curses Subroutines** ..................................................... 573
addch, mvaddch, mvwaddch, or waddch Subroutine .................. 573
addnstr, addstr, mvaddnstr, mvaddstr, mwaddstr, waddnstr, or waddstr Subroutine .................. 574
attroff, atrton, attrset, wattroff, wattron, or wattrset Subroutine .................. 576
attron or wattron Subroutine ................................................. 578
attrset or wattrset Subroutine ................................................. 578
baudrate Subroutine .............................................................. 579
beep Subroutine ................................................................. 580
box Subroutine ........................................................................ 581
can_change_color, color_content, has_colors, init_color, init_pair, start_color or pair_content Subroutine .......................................................... 582
cbreak, nocbreak, noraw, or raw Subroutine ............................................. 585
clear, erase, wclear or werase Subroutine ............................................. 586
clearok, idlok, leaveok, scrollok, setscrreg or wsetscrreg Subroutine .................. 587
clrtoeol or wcrtomb Subroutine .................................................... 590
clearwin Subroutine ................................................................ 591
copywin Subroutine .................................................................. 592
curs_set Subroutine ................................................................ 593
def_prog_mode, def_shell_mode, reset_prog_mode or reset_shell_mode Subroutine .......................... 595
def_shell_mode Subroutine ......................................................... 596
delcurterm, restartterm, setcurterm, or setupterm Subroutine .................. 597
delay_output Subroutine ............................................................ 599
delch, mvdelch, mvwdelch or wdelch Subroutine .................. 600
deleteln or wdeleteln Subroutine ................................................. 601
delwin Subroutine .................................................................. 602
echo or noecho Subroutine .......................................................... 602
echochar or wechochar Subroutines .................................................. 603
eendwin Subroutine .................................................................. 604
erase or werase Subroutine .......................................................... 605
erasechar, erasedchar, killchar, and killwchar Subroutine .................. 606
filter Subroutine ...................................................................... 607
flash Subroutine ....................................................................... 607
flushinp Subroutine ................................................................ 608
garbage Subroutine .................................................................. 609
getbegyx, getmaxyx, getparyx, or getyx Subroutine .................. 610
getch, mvgetch, mvwgetch, or wgetch Subroutine .................. 611
getmaxyx Subroutine ................................................................ 615
getnstr, getstr, mvgetnstr, mvgetstr, mvwgetnstr, mwgetstr, wgetnstr, or wgetstr Subroutine .................. 616
getsyx Subroutine .................................................................... 618
getyx Macro ........................................................................... 619
halfdelay Subroutine ................................................................ 620
has_colors Subroutine .............................................................. 620
has_ic and has_il Subroutine ....................................................... 621
has_il Subroutine ................................................................. 622
idlok Subroutine ...................................................................... 623
inch, mvinch, mvwinch, or winch Subroutine .................. 624
init_color Subroutine .............................................................. 625
init_pair Subroutine ................................................................ 626
initscr and newterm Subroutine .................................................. 627
insch, mvinsch, mvwinsch, or winsch Subroutine
insertln or winsertln Subroutine
intrflush Subroutine
keyname, key_name Subroutine
keypad Subroutine
killchar or killwchar Subroutine
_lazySetErrorHandler Subroutine
leaveok Subroutine
longname Subroutine
makenew Subroutine
meta Subroutine
move or wmove Subroutine
mvcur Subroutine
mvwin Subroutine
newpad, pnoutrefresh, prefresh, or subpad Subroutine
newterm Subroutine
derwin, newwin, or subwin Subroutine
nl or nonl Subroutine
noDelay Subroutine
notimeout, timeout, wtimeout Subroutine
overlay or overwrite Subroutine
pair_content Subroutine
printw, wprintw, mvprintw, or mvwprintw Subroutine
putp, tputs Subroutine
raw or noraw Subroutine
refresh or wrefresh Subroutine
reset_prog_mode Subroutine
reset_shell_mode Subroutine
resetterm Subroutine
resetty, savetty Subroutine
restartterm Subroutine
scr_dump, scr_init, scr_restore, scr_set Subroutine
scr_init Subroutine
scr_restore Subroutine
scrl, scroll, wscrl Subroutine
scrollok Subroutine
set_curterm Subroutine
setscter or wsetscter Subroutine
setsyx Subroutine
set_term Subroutine
setupterm Subroutine
_showstring Subroutine
slk_attroff, slk_attr_off, slk_attron, slk_attrset, slk_attr_set, slk_clear, slk_color, slk_init, slk_label,
slk_noutrefresh, slk_refresh, slk_restore, slk_set, slk_touch, slk_wset, Subroutine
slk_init Subroutine
slk_label Subroutine
slk_noutrefresh Subroutine
slk_refresh Subroutine
slk_restore Subroutine
slk_set Subroutine
slk_touch Subroutine
standend, standout, wstandend, or wstandout Subroutine
Chapter 3. FORTRAN Basic Linear Algebra Subroutines (BLAS)

SDOT or DDOT Function ................................................. 711
CDOTC or ZDOTC Function .............................................. 711
CDOTU or ZDOTU Function .............................................. 712
SAXPY, DAXPY, CAXPY, or ZAXPY Subroutine .......................... 713
SROTG, DROTG, CROTG, or ZROTG Subroutine .......................... 713
SROT, DROT, CSROT, or ZDROT Subroutine ............................ 714
SCOPY, DCOPY, CCOPY, or ZCOPY Subroutine .......................... 715
SSWAP, DSWAP, CSWAP, or ZSWAP Subroutine .......................... 716
SNRM2, DNRM2, SCNRM2, or DZNRM2 Function .......................... 717
SASUM, DASUM, SCASUM, or DZASUM Function .......................... 718
SSCAL, DSCAL, CSSCAL, CSCAL, ZDSCAL, or ZSCAL Subroutine ......... 718
ISAMAX, IDAMAX, ICAMAX, or IZAMAX Function ........................ 719
SDSDOT Function ....................................................... 720
SROTM or DROTM Subroutine ............................................ 721
SROTMG or DROTMG Subroutine ........................................... 722
SGEMV, DGEMV, CGEMV, or ZGEMV Subroutine .......................... 723
SGBMV, DGBMV, CGBMV, or ZGBMV Subroutine ........................ 724
CHEMV or ZHEMV Subroutine ............................................ 726
CHBMV or ZHBMV Subroutine ............................................ 727
CHPMV or ZHPMV Subroutine ............................................ 729
SSYMV or DSYMV Subroutine ............................................ 730
SSBMV or DSBMV Subroutine ............................................ 731
SSPMV or DSPMV Subroutine ............................................ 732
STRMV, DTRMV, CTRMV, or ZTRMV Subroutine .......................... 733
STBMV, DTBMV, CTBMV, or ZTBVMV Subroutine ........................ 735
STPMV, DTPMV, CTPMV, or ZTPMV Subroutine .......................... 737
STRSV, DTRSV, CTRSV, or ZTRSV Subroutine .......................... 738
STBSV, DTBSV, CTBSV, or ZTBSV Subroutine .......................... 740
STPSV, DTPSV, CTPSV, or ZTPSV Subroutine .......................... 742
SGER or DGER Subroutine ................................................. 744
CGERU or ZGERU Subroutine .............................................. 745
CGERC or ZGERC Subroutine .............................................. 745
CHER or ZHER Subroutine ................................................. 746
CHPR or ZHPR Subroutine .................................................. 747
CHER2 or ZHER2 Subroutine ................................................ 748
CHPR2 or ZHPR2 Subroutine ................................................. 749
SSYR or DSYSR Subroutine .................................................. 750
SSPR or DSPR Subroutine .................................................... 751
SSYR2 or DSPY2 Subroutine ............................................... 752
SSPR2 or DSPR2 Subroutine ................................................. 754
SGEMM, DGEMM, CGEMM, or ZGEMM Subroutine .................... 755
SSYMM, DSYMM, CSYM, or ZSYM Subroutine ......................... 756
CHEMM or ZHEMM Subroutine ............................................. 758
SSYRK, DSYRK, CSYRK, or ZSYRK Subroutine ......................... 760
CHERK or ZHERK Subroutine .............................................. 761
SSYR2K, DSYR2K, CSYR2K, or ZSYR2K Subroutine ................. 763
CHER2K or ZHER2K Subroutine ........................................... 764
STRMM, DTRMM, CTRMM, or ZTRMM Subroutine ..................... 766
STRSM, DTRSM, CTRSM, or ZTRSM Subroutine ....................... 768

Appendix A. Base Operating System Error Codes for Services That Require Path-Name Resolution .................................................. 771

Appendix B. ODM Error Codes ................................................. 773

Appendix C. Notices ................................................................ 775
Trademarks ............................................................................ 776

Index ..................................................................................... 777
About This Book

This book provides experienced C programmers with complete detailed information about Base Operating System runtime services for the AIX® operating system. Runtime services are listed alphabetically, and complete descriptions are given for them. This volume contains AIX services that begin with the letters Q through Z. To use the book effectively, you should be familiar with commands, system calls, subroutines, file formats, and special files. This publication is also available on the documentation CD that is shipped with the operating system.

This book is part of the six-volume technical reference set, *AIX 5L Version 5.3 Technical Reference*, that provides information on system calls, kernel extension calls, and subroutines in the following volumes:

- *AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 1* and *AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 2* provide information on system calls, subroutines, functions, macros, and statements associated with base operating system runtime services.

- *AIX 5L Version 5.3 Technical Reference: Communications Volume 1* and *AIX 5L Version 5.3 Technical Reference: Communications Volume 2* provide information on entry points, functions, system calls, subroutines, and operations related to communications services.

- *AIX 5L Version 5.3 Technical Reference: Kernel and Subsystems Volume 1* and *AIX 5L Version 5.3 Technical Reference: Kernel and Subsystems Volume 2* provide information about kernel services, device driver operations, file system operations, subroutines, the configuration subsystem, the communications subsystem, the low function terminal (LFT) subsystem, the logical volume subsystem, the M-audio capture and playback adapter subsystem, the printer subsystem, the SCSI subsystem, and the serial DASD subsystem.

Highlighting

The following highlighting conventions are used in this book:

- **Bold** Identifies commands, subroutines, keywords, files, structures, directories, and other items whose names are predefined by the system. Also identifies graphical objects such as buttons, labels, and icons that the user selects.

- **Italics** Identifies parameters whose actual names or values are to be supplied by the user.

- **Monospace** Identifies examples of specific data values, examples of text similar to what you might see displayed, examples of portions of program code similar to what you might write as a programmer, messages from the system, or information you should actually type.

Case-Sensitivity in AIX

Everything in the AIX operating system is case-sensitive, which means that it distinguishes between uppercase and lowercase letters. For example, you can use the `ls` command to list files. If you type `LS`, the system responds that the command is “not found.” Likewise, `FILEA`, `FiLea`, and `filea` are three distinct file names, even if they reside in the same directory. To avoid causing undesirable actions to be performed, always ensure that you use the correct case.

ISO 9000

ISO 9000 registered quality systems were used in the development and manufacturing of this product.
32-Bit and 64-Bit Support for the Single UNIX Specification

Beginning with Version 5.2, the operating system is designed to support The Open Group’s Single UNIX Specification Version 3 (UNIX 03) for portability of UNIX-based operating systems. Many new interfaces, and some current ones, have been added or enhanced to meet this specification, making Version 5.2 even more open and portable for applications, while remaining compatible with previous releases of AIX. To determine the proper way to develop a UNIX 03-portable application, you may need to refer to The Open Group’s UNIX 03 specification, which can be accessed online or downloaded from http://www.unix.org/.

Related Publications

The following books contain information about or related to application programming interfaces:

- AIX 5L Version 5.3 System Management Guide: Operating System and Devices
- AIX 5L Version 5.3 System Management Guide: Communications and Networks
- AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs
- AIX 5L Version 5.3 Communications Programming Concepts
- AIX 5L Version 5.3 Kernel Extensions and Device Support Programming Concepts
- AIX 5L Version 5.3 Files Reference
Chapter 1. Base Operating System (BOS) Runtime Services (Q-Z)

qsort Subroutine

Purpose
Sorts a table of data in place.

Library
Standard C Library (libc.a)

Syntax
```
#include <stdlib.h>

void qsort (Base, NumberOfElements, Size, ComparisonPointer)
```

Description
The qsort subroutine sorts a table of data in place. It uses the quicker-sort algorithm.

Parameters
- **Base**: Points to the element at the base of the table.
- **NumberOfElements**: Specifies the number of elements in the table.
- **Size**: Specifies the size of each element.
- **ComparisonPointer**: Points to the comparison function, which is passed two parameters that point to the objects being compared. The qsort subroutine sorts the array in ascending order according to the comparison function.

Return Values
The comparison function compares its parameters and returns a value as follows:
- If the first parameter is less than the second parameter, the ComparisonPointer parameter returns a value less than 0.
- If the first parameter is equal to the second parameter, the ComparisonPointer parameter returns 0.
- If the first parameter is greater than the second parameter, the ComparisonPointer parameter returns a value greater than 0.

Because the comparison function need not compare every byte, the elements can contain arbitrary data in addition to the values being compared.

**Note**: If two items are the same when compared, their order in the output of this subroutine is unpredictable.

The pointer to the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.
Related Information

The `bsearch` subroutine, `lsearch` subroutine.

Searching and Sorting Example Program | Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

quotactl Subroutine

Purpose
Manipulates disk quotas.

Library
Standard C Library (libc.a)

Syntax

```c
#include <jfs/quota.h>

int quotactl (Path, Cmd, ID, Addr)

int Cmd, ID;
char *Addr, *Path;
```

Description

The `quotactl` subroutine enables, disables, and manipulates disk quotas for file systems on which quotas have been enabled.

On AIX, disk quotas are supported by the legacy Journaled File System (JFS) and the enhanced Journaled File System (JFS2).

The `Cmd` parameter is constructed through use of the `QCMD(Qcmd, type)` macro contained within the `sys/fs/quota_common.h` file. The `Qcmd` parameter specifies the quota control command. The `type` parameter specifies either user (USRQUOTA) or group (GRPQUOTA) quota type.

The valid values for the `Cmd` parameter in all supported file system types are:

**Q_QUOTAON**

Enables disk quotas for the file system specified by the `Path` parameter. The `Addr` parameter specifies a file from which to take the quotas. The quota file must exist; it is normally created with the `quotacheck` command. The `ID` parameter is unused. Root user authority is required to enable quotas. By specifying the new quota file path in the `Addr` parameter, the `quotactl` command can also be used to change the quota file that is being used without first disabling disk quotas.

**Q_QUOTAOFF**

Disables disk quotas for the file system specified by the `Path` parameter. The `Addr` and `ID` arguments are unused. Root user authority is required to disable quotas.

Additional JFS specific values for the `Cmd` parameter are as follows:

**Q_GETQUOTA**

Gets disk quota limits and current usage for a user or group specified by the `ID` parameter. The `Addr` parameter points to a `dqblk` buffer to hold the returned information. The `dqblk` structure is defined in the `jfs/quota.h` file. Root user authority is required if the `ID` value is not the current ID of the caller.

**Q_SETQUOTA**

Sets disk quota limits for the user or group specified by the `ID` parameter. The `Addr` parameter
points to a `dqblk` buffer containing the new quota limits. The `dqblk` structure is defined in the `jfs/quota.h` file. Root user authority is required to set quotas.

Q_SETUSE

Sets disk usage limits for the user or group specified by the `ID` parameter. The `Addr` parameter points to a `dqblk` buffer containing the new usage limits. The `dqblk` structure is defined in the `jfs/quota.h` file. Root user authority is required to set disk usage limits.

Additional JFS2 specific values for the `Cmd` parameter are as follows:

Q_J2GETQUOTA

Gets quota limits, current usage, and time remaining in grace periods for the user or group specified by the `ID` parameter. The `Addr` parameter points to a `quota64_t` buffer to hold the returned information. The `quota64_t` structure is defined in the `quota_common.h` file. Root user authority is required if the `ID` value is not the current ID of the caller.

Q_J2PUTQUOTA

Updates (replaces) the current usage values for the user or group specified by the `ID` parameter. The `Addr` parameter points to a `quota64_t` buffer holding the new information. The `quota64_t` structure is defined in the `quota_common.h` file. Root user authority is required.

Q_J2GETLIMIT

Gets quota limits information for the Limits Class specified by the `ID` parameter. The `Addr` parameter points to a `j2qlimit_t` buffer to hold the returned information. The `j2qlimit_t` structure is defined in the `j2/j2_quota.h` file. Root user authority is required.

Q_J2PUTLIMIT

Updates quota limits information for the Limits Class specified by the `ID` parameter. The `Addr` parameter points to a `j2qlimit_t` buffer holding the new information. The `j2qlimit_t` structure is defined in the `j2/j2_quota.h` file. Root user authority is required.

Q_J2NEWLIMIT

Creates a new Limits Class and updates it with the quota limits information from `Addr`. The `ID` parameter is ignored. The `Addr` parameter points to a `j2qlimit_t` buffer holding the new information. The `j2qlimit_t` structure is updated with the new Limits Class ID and returned to the user. The `j2qlimit_t` structure is defined in the `j2/j2_quota.h` file. Root user authority is required.

Q_J2RMVLIMIT

Marks the Limits Class specified by the `ID` parameter as deleted. Any Usage record referencing a deleted Limits Class is now limited by the default Limits Class. The `Addr` parameter is ignored. Root user authority is required.

Q_J2DEFLIMIT

Sets the Limits Class specified by the `ID` parameter as the default Limits Class. The `Addr` parameter is ignored. Root user authority is required.

Q_J2USELIMIT

Binds a Usage record to the Limits Class specified by the `ID` parameter. The Limits Class must be valid; otherwise, `ENOENT` is returned. Use the `Addr` parameter to pass a pointer to the user ID or group ID. Root user authority is required.

Q_J2GETNEXTQ

Returns the ID of the next allocated, nondeleted Limits Class higher than the ID specified by the `ID` parameter. The `Addr` parameter points to a buffer containing a `uid_t` structure. Root user authority is required.

Q_J2INITFILE

Initializes an existing quota file. The `Addr` and `ID` parameters are ignored. Root user authority is required.

Q_J2QUOTACK

Performs a consistency check on an existing quota file. If any of the control data within the file is
invalid or inconsistent, **Q_J2QUOTACHK** attempts to reconstruct the control data based on existing quota data in the file. If no **qquota** data can be recognized, the file is initialized. The **Addr** and **ID** parameters are ignored. Root user authority is required.

### Parameters

**Path**
- Specifies the path name of any file within the mounted file system to which the quota control command is to be applied. Typically, this would be the mount point of the file system.

**Cmd**
- Specifies the quota control command to be applied and whether it is applied to a user or group quota.

**ID**
- Specifies the user or group ID to which the quota control command applies. The **ID** parameter is interpreted by the specified quota type. The JFS file system supports quotas for IDs within the range of **MINDQID** through **MAXDQID**; JFS2 supports all IDs.

**Addr**
- Points to the address of an optional, command-specific, data structure that is copied in or out of the system. The interpretation of the **Addr** parameter for each quota control command is given above.

### Return Values

A successful call returns 0; otherwise, the value -1 is returned and the **errno** global variable indicates the reason for the failure.

### Error Codes

A **quotactl** subroutine will fail when one of the following occurs:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EACCES</td>
<td>In the <strong>Q_QUOTAON</strong> command, the quota file is not a regular file.</td>
</tr>
<tr>
<td>EACCES</td>
<td>Search permission is denied for a component of a path prefix.</td>
</tr>
<tr>
<td>EFAULT</td>
<td>An invalid <strong>Addr</strong> parameter is supplied; the associated structure could not be copied in or out of the kernel.</td>
</tr>
<tr>
<td>EFAULT</td>
<td>The <strong>Path</strong> parameter points outside the process’s allocated address space.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>The specified quota control command or quota type is invalid.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>Path name contains a character with the high-order bit set.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>The <strong>ID</strong> parameter is outside of the supported range of <strong>MINDQID</strong> through <strong>MAXDQID</strong> (JFS only).</td>
</tr>
<tr>
<td>EIO</td>
<td>An I/O error occurred while reading or writing the quotas file.</td>
</tr>
<tr>
<td>ELOOP</td>
<td>Too many symbolic links were encountered in translating a path name.</td>
</tr>
<tr>
<td>ENAMETOOLONG</td>
<td>A component of either path name exceeded 255 characters, or the entire length of either path name exceeded 1023 characters.</td>
</tr>
<tr>
<td>ENOENT</td>
<td>A file name does not exist.</td>
</tr>
<tr>
<td>ENOTBLK</td>
<td>Mounted file system is not a block device.</td>
</tr>
<tr>
<td>ENOTDIR</td>
<td>A component of a path prefix is not a directory.</td>
</tr>
<tr>
<td>EOPNOTSUPP</td>
<td>The file system does not support quotas.</td>
</tr>
<tr>
<td>EPERM</td>
<td>The quota control commands is privileged and the caller did not have root user authority.</td>
</tr>
<tr>
<td>EROFS</td>
<td>In the <strong>Q_QUOTAON</strong> command, the quota file resides on a read-only file system.</td>
</tr>
<tr>
<td>EUSERS</td>
<td>The in-core quota table cannot be expanded (JFS only).</td>
</tr>
</tbody>
</table>

### Related Information

- The **quotacheck** command.

---

4 Technical Reference, Volume 2: Base Operating System and Extensions
raise Subroutine

Purpose
Sends a signal to the currently running program.

Libraries
Standard C Library (libc.a)
Threads Library (libpthreads.a)

Syntax
#include <sys/signal.h>

int raise (Signal)
int Signal;

Description
The raise subroutine sends the signal specified by the Signal parameter to the executing process or thread, depending if the POSIX threads API (the libpthreads.a library) is used or not. When the program is not linked with the threads library, the raise subroutine sends the signal to the calling process as follows:
return kill(getpid(), Signal);

When the program is linked with the threads library, the raise subroutine sends the signal to the calling thread as follows:
return pthread_kill(pthread_self(), Signal);

When using the threads library, it is important to ensure that the threads library is linked before the standard C library.

Parameter
Signal Specifies a signal number.

Return Values
Upon successful completion of the raise subroutine, a value of 0 is returned. Otherwise, a nonzero value is returned, and the errno global variable is set to indicate the error.

Error Code
EINVAL The value of the sig argument is an invalid signal number

Related Information
The exit subroutine, kill subroutine, pthread_kill subroutine, sigaction subroutine, sigaction, sigvec, or signal subroutine.

Signal Management in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs provides more information about signal management in multi-threaded processes.
rand or srand Subroutine

Purpose

Generates pseudo-random numbers.

Library

Standard C Library (libc.a)

Syntax

```c
#include <stdlib.h>

int rand

void srand (Seed);

unsigned int Seed;
```

Description

**Attention:** Do not use the `rand` subroutine in a multithreaded environment. See the multithread alternative in the `rand_r` subroutine article.

The `rand` subroutine generates a pseudo-random number using a multiplicative congruential algorithm. The random-number generator has a period of 2**32, and it returns successive pseudo-random numbers in the range from 0 through (2**15) -1.

The `srand` subroutine resets the random-number generator to a new starting point. It uses the `Seed` parameter as a seed for a new sequence of pseudo-random numbers to be returned by subsequent calls to the `rand` subroutine. If you then call the `srand` subroutine with the same seed value, the `rand` subroutine repeats the sequence of pseudo-random numbers. When you call the `rand` subroutine before making any calls to the `srand` subroutine, it generates the same sequence of numbers that it would if you first called the `srand` subroutine with a seed value of 1.

**Note:** The `rand` subroutine is a simple random-number generator. Its spectral properties, a mathematical measurement of randomness, are somewhat limited. See the `drand48` subroutine or the `random` subroutine for more elaborate random-number generators that have greater spectral properties.

Parameter

`Seed` Specifies an initial seed value.

Return Values

Upon successful completion, the `rand` subroutine returns the next random number in sequence. The `srand` subroutine returns no value.

There are better random number generators, as noted above; however, the `rand` and `srand` subroutines are the interfaces defined for the ANSI C library.

Example

The following functions define the semantics of the `rand` and `srand` subroutines, and are included here to facilitate porting applications from different implementations:

```c
static unsigned int next = 1;
int rand( )
{
```
next = next
* 1103515245 + 12345;
return ((next >>16) & 32767);
}
void srand (Seed)

unsigned
int Seed;
{
next = Seed;
}

Related Information
The [drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, or lcong48] subroutine, 
random, srandom, initstate, or setstate subroutine. See the 
random, srandom, initstate, or setstate Subroutine on page 8
subroutine.

Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging
Programs.

rand_r Subroutine

Purpose
Generates pseudo-random numbers.

Libraries
Thread-Safe C Library (libc_r.a)
Berkeley Compatibility Library (libbsd.a)

Syntax
#include <stdlib.h>

int rand_r (Seed)
unsigned int * Seed;

Description
The rand_r subroutine generates and returns a pseudo-random number using a multiplicative congruential
algorithm. The random-number generator has a period of 2**32, and it returns successive pseudo-random
numbers.

Note: The rand_r subroutine is a simple random-number generator. Its spectral properties (the
mathematical measurement of the randomness of a number sequence) are limited. See the 
drand48 subroutine or the random subroutine for more elaborate random-number generators that have greater spectral properties.

Programs using this subroutine must link to the libpthreads.a library.

Parameter

Seed Specifies an initial seed value.
Return Values

0    Indicates that the subroutines was successful.
-1   Indicates that the subroutines was not successful.

Error Codes

If the following condition occurs, the rand_r subroutine sets the errno global variable to the corresponding value.

EINVAL        The Seed parameter specifies a null value.

File

/usr/include/sys/types.h               Defines system macros, data types, and subroutines.

Related Information

The drand48 subroutine, random subroutine, random, srandom, initstate, or setstate Subroutine.

Subroutines Overview and List of Multithread Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

random, srandom, initstate, or setstate Subroutine

Purpose

Generates pseudo-random numbers more efficiently.

Library

Standard C Library (libc.a)

Syntax

#include <stdlib.h>

long random ( )
void srandom (Seed)
unsigned int Seed;

char *initstate (Seed, State, Number)
unsigned int Seed;
char *State;
size_t Number;
char *setstate (State)
const char *State;

Description

Attention:  Do not use the random, srandom, initstate, or setstate subroutine in a multithreaded environment.

The random subroutine uses a non-linear 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 very large, 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 a full 256 bytes of state information, the period of the random-number generator is greater than $2^{69}$, which should be sufficient for most purposes.

The `rand` and `srand` subroutines have almost the same calling sequence and initialization properties as the `rand` and `srand` subroutines. The difference is that the `rand` subroutine produces a much less random sequence; in fact, the low dozen bits generated by the `rand` subroutine go through a cyclic pattern. All the bits generated by the `random` subroutine are usable. For example, `random()` & 01 produces a random binary value.

The `srand` subroutine, unlike the `srand` subroutine, does not return the old seed because the amount of state information used is more than a single word. The `initstate` subroutine and `setstate` subroutine handle restarting and changing random-number generators. Like the `rand` subroutine, however, the `random` subroutine by default produces a sequence of numbers that can be duplicated by calling the `srandom` subroutine with 1 as the seed.

The `initstate` subroutine allows a state array, passed in as an argument, to be initialized for future use. The size of the state array (in bytes) is used by the `initstate` subroutine, to decide how sophisticated a random-number generator it should use; the larger the state array, the more random are the numbers. Values for the amount of state information are 8, 32, 64, 128, and 256 bytes. For amounts greater than or equal to 8 bytes, or less than 32 bytes, the `random` subroutine uses a simple linear congruential random number generator, while other amounts are rounded down to the nearest known value. The `Seed` parameter specifies a starting point for the random-number sequence and provides for restarting at the same point. The `initstate` subroutine returns a pointer to the previous state information array.

Once a state has been initialized, the `setstate` subroutine allows rapid switching between states. The array defined by `State` parameter is used for further random-number generation until the `initstate` subroutine is called or the `setstate` subroutine is called again. The `setstate` subroutine returns a pointer to the previous state array.

After initialization, a state array can be restarted at a different point in one of two ways:

- The `initstate` subroutine can be used, with the desired seed, state array, and size of the array.
- The `setstate` subroutine, with the desired state, can be used, followed by the `srandom` subroutine with the desired seed. The advantage of using both of these subroutines is that the size of the state array does not have to be saved once it is initialized.

**Parameters**

- `Seed` Specifies an initial seed value.
- `State` Points to the array of state information.
- `Number` Specifies the size of the state information array.

**Error Codes**

If the `initstate` subroutine is called with less than 8 bytes of state information, or if the `setstate` subroutine detects that the state information has been damaged, error messages are sent to standard error.

**Related Information**

The `drand48`, `erand48`, `jrand48`, `lcong48`, `lrand48`, `mrand48`, `nrand48`, `seed48`, or `srand48` subroutine, `rand` or `srand` subroutine.
ra_attachrset Subroutine

Purpose
Attaches a work component to a resource set.

Library
Standard C library (libc.a)

Syntax
```
#include <sys/rset.h>
int ra_attachrset (rstype, rsid, rset, flags)
rstype_t rstype;
rsid_t rsid;
rsethandle_t rset;
unsigned int flags;
```

Description
The `ra_attachrset` subroutine attaches a work component specified by the `rstype` and `rsid` parameters to a resource set specified by the `rset` parameter.

The work component is an existing process identified by the process ID or an existing kernel thread identified by the kernel thread ID (tid). A process ID or thread ID value of RS_MYSELF indicates the attachment applies to the current process or the current kernel thread, respectively.

The following conditions must be met to successfully attach a process to a resource set:

- The resource set must contain processors that are available in the system.
- The calling process must either have root authority or have CAP_NUMA_ATTACH capability.
- The calling process must either have root authority or the same effective userid as the target process.
- The target process must not contain any threads that have bindprocessor bindings to a processor.
- The resource set must be contained in (be a subset of) the target process' partition resource set.
- The resource set must be a superset of all the threads' `rset` in the target process.
- For R_FILDES `rstype`, the calling process must specify an open file descriptor, and it must have write access to the file, or the calling process' effective userid must be equal to the file owner's userid.
- For R_SHM `rstype`, the calling process' effective userid must be equal to the shared segment's owner.

The following conditions must be met to successfully attach a kernel thread to a resource set:

- The resource set must contain processors that are available in the system.
- The calling process must either have root authority or have CAP_NUMA_ATTACH capability.
- The calling process must either have root authority or the same effective userid as the target process.
- The target thread must not have bindprocessor bindings to a processor.
- The resource set must be contained in (be a subset of) the target thread's process effective and partition resource set.

If any of these conditions are not met, the attachment will fail.

Once a process is attached to a resource set, the threads in the process will only run on processors contained in the resource set. Once a kernel thread is attached to a resource set, the threads will only run on processors contained in the resource set.
Dynamic Processor Deallocation and DLPAR may invalidate the processor attachment that is being specified. A program must become DLPAR Aware to resolve this problem.

The flags parameter can be set to indicate the policy for using the resources contained in the resource set specified in the rset parameter. The only supported policy today is RS_STRSET, which is useful only when the processors of the system are running in simultaneous multi-threading mode. Processors like the POWER5 support simultaneous multi-threading, where each physical processor has two execution engines, called hardware threads. Each hardware thread is essentially equivalent to a single CPU, and each is identified as a separate CPU in a resource set. The RS_STRSET flag indicates that the process is to be scheduled with a single-threaded policy; namely, that it should be scheduled on only one hardware thread per physical processor. All the available processors indicated in the resource set are required to specify this flag for exclusive use. A new resource set, called an ST resource set, is constructed from the specified resource set and attached to the process according to the following rules:

- All offline processors are ignored.
- If all the hardware threads (CPUs) of a physical processor (when running in simultaneous multi-threading mode, there will be more than one active hardware thread per physical processor) are not included in the specified resource set, the other CPUs of the processor are ignored when constructing the ST resource set.
- Only one CPU (hardware thread) resource per physical processor is included in the ST resource set.

**Parameters**

- **rstype**: Specifies the type of work component to be attached to the resource set specified by the rset parameter. The rstype parameter must be the following value, defined in rset.h:
  - R_PROCESS: existing process
  - R_THREAD: existing kernel thread
  - R_FILDES: file identified by an open file descriptor
  - R_SHM: shared memory segment identified by shared memory segment ID
  - R_SUBRANGE: attachment involves a subrange of the work component

- **rsid**: Identifies the work component to be attached to the resource set specified by the rset parameter. The rsid parameter must be the following:
  - Process ID (for rstype of R_PROCESS): set the rsid_t at_pid field to the desired process' process ID.
  - Kernel thread ID (for rstype of R_THREAD): set the rsid_t at_tid field to the desired kernel thread's thread ID.
  - Open file descriptor (for rstype of R_FILDES): set the rsid_t at_fd field to the desired file descriptor.
  - Shared memory segment ID (for rstype of R_SHM): set the rsid_t at_shmid field to the desired shared memory ID.
  - Pointer to a subrange_t struct (for rstype of R_SUBRANGE): set the subrange_t su_offset, su_length, su_rstype, and su_rsid fields. The other fields in the subrange_t struct are ignored. The memory allocation policy is taken from the flags parameter, not the su_policy field.

- **rset**: Specifies which work component (specified by the rstype and rsid parameters) to attach to the resource set.
- **flags**: Specifies memory allocation policy for the resource being attached.
  - R_DEFAULT: Default memory policy
  - R_FIRST_TOUCH: First access memory policy
  - R_BALANCED: Balanced memory policy

If the rstype parameter is R_SUBRANGE, the memory allocation policy is specified in the subrange_t su_policy field rather than in the flags parameter. A value of RS_STRSET indicates that the process is to be scheduled with a single-threaded policy (only on one hardware thread per physical processor).

**Return Values**

If successful, a value of 0 is returned. If unsuccessful, a value of -1 is returned and the errno global variable is set to indicate the error.
Error Codes
The ra_attachrset subroutine is unsuccessful if one or more of the following are true:

EINVAL One of the following is true:
- The flags parameter contains an invalid value.
- The rstype parameter contains an invalid type qualifier.
- The RS_STRSET flags parameter is specified and one or more processors in the rset parameter are not assigned for exclusive use.

ENODEV The resource set specified by the rset parameter does not contain any available processors, or the RS_STRSET flags parameter is specified and the constructed ST resource set does not have any available processors.

ESRCH The process or kernel thread identified by the rstype and rsid parameters does not exist.

EPERM One of the following is true:
- If the rstype is R_PROCESS, either the resource set specified by the rset parameter is not included in the partition resource set of the process identified by the rstype and rsid parameters, or any of the thread's R_THREAD rset in this process is not a subset of the resource set specified by the rset parameter.
- If the rstype is R_THREAD, the resource set specified by the rset parameter is not included in the target thread's process effective or partition (real) resource set.
- The calling process has neither root authority nor CAP_NUMA_ATTACH attachment privilege.
- The calling process has neither root authority nor the same effective user ID as the process identified by the rstype and rsid parameters.
- The process or thread identified by the rstype and rsid parameters has one or more threads with a bindprocessor processor binding.

Related Information
“ra_fork Subroutine” on page 16, “ra_exec Subroutine” on page 14, “ra_getrset Subroutine” on page 21, and “ra_detachrset Subroutine.”

The Dynamic Logical Partitioning article in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

The dr_reconfig system call in AIX 5L Version 5.3 Technical Reference: Kernel and Subsystems Volume 1.

ra_detachrset Subroutine

Purpose
Detaches a work component from a resource set.

Library
Standard C library (libc.a)

Syntax
#include <sys/rset.h>

int ra_detachrset (rstype, rsid, flags)

rstype_t rstype;
rsid_t rsid;
unsigned int flags;

Description
The ra_detachrset subroutine detaches a work component specified by rstype and rsid from a resource set.
The work component is an existing process identified by the process ID or an existing kernel thread identified by the kernel thread ID (tid). A process ID or thread ID value of RS_MYSELF indicates the detach command applies to the current process or the current kernel thread, respectively.

The following conditions must be met to detach a process or a kernel thread from a resource set:

- The calling process must either have root authority or have CAP_NUMA_ATTACH capability.
- The calling process must either have root authority or the same effective userid as the target process.
- For R_FILDES rstype, the calling process must specify an open file descriptor, and it must have write access to the file, or the calling process' effective userid must be equal to the file owner's userid.
- For R_SHM rstype, the calling process' effective userid must be equal to the shared segment's owner.

If these conditions are not met, the operation will fail.

Once a process is detached from a resource set, the threads in the process can run on all available processors contained in the process' partition resource set. Once a kernel thread is detached from a resource set, that thread can run on all available processors contained in its process effective or partition resource set.

**Parameters**

- **rstype**
  Specifies the type of work component to be detached from to the resource set specified by rset. This parameter must be the following value, defined in rset.h:
  - R_PROCESS: existing process
  - R_THREAD: existing kernel thread
  - R_FILDES: file identified by an open file descriptor
  - R_SHM: shared memory segment identified by shared memory segment ID
  - R_SUBRANGE: attachment involves a subrange of the work component

- **rsid**
  Identifies the work component to be attached to the resource set specified by rset. This parameter must be the following:
  - Process ID (for rstype of R_PROCESS): set the rsid_t at_pid field to the desired process' process ID.
  - Kernel thread ID (for rstype of R_THREAD): set the rsid_t at_tid field to the desired kernel thread's thread ID.
  - Open file descriptor (for rstype of R_FILDES): set the rsid_t at_fd field to the desired file descriptor.
  - Shared memory segment ID (for rstype of R_SHM): set the rsid_t at_shmid field to the desired shared memory ID.
  - Pointer to a subrange_t struct (for rstype of R_SUBRANGE): set the subrange_t su_offset, su_length, su_rstype, and su_rsid fields. The other fields in the subrange_t struct are ignored.

- **flags**
  For rstype of R_PROCESS, the R_DETACH_ALLTHRDS indicates that R_THREAD rsets are detached from all threads in a specified process. The process' effective rset is not detached in this case. Reserved for future use. Specify as 0.

**Return Values**

If successful, a value of 0 is returned. If unsuccessful, a value of -1 is returned, and the errno global variable is set to indicate the error.

**Error Codes**

The ra_detachrset subroutine is unsuccessful if one or more of the following are true:

- **EINVAL**
  One of the following is true:
  - The flags parameter contains an invalid value.
  - The rstype parameter contains an invalid type qualifier.

- **ESRCH**
  The process or kernel thread identified by the rstype and rsid parameters does not exist.
One of the following is true:

- The calling process has neither root authority nor CAP_NUMA_ATTACH attachment privilege.
- The calling process has neither root authority nor the same effective user ID as the process identified by the `rstype` and `rsid` parameters.

**Related Information**


**ra_exec Subroutine**

**Purpose**

Executes a file and attaches it to a given resource.

**Library**

Standard C library (`libc.a`)

**Syntax**

```c
#include <sys/rset.h>

int ra_exec(rstype_t rstype, rsid_t rsid, flags_t flags, path, argument0 [,argument1,...], 0)
int ra_execle(rstype_t rstype, rsid_t rsid, flags_t flags, path, argument0[,argument1,...], 0, envptr)
int ra_execlep(rstype_t rstype, rsid_t rsid, flags_t flags, File, argument0[,argument1,...], 0)
int ra_execv(rstype_t rstype, rsid_t rsid, flags_t flags, path, argumentv)
int ra_execve(rstype_t rstype, rsid_t rsid, flags_t flags, path, argumentv, envptr)
int ra_execvp(rstype_t rstype, rsid_t rsid, flags_t flags, File, argumentv)
```

14 Technical Reference, Volume 2: Base Operating System and Extensions
int ra_exec(rstype, rsid, flags, path, argumentv[], envptr)
rtype_t rstype;
rsid_t rsid;
unsigned int flags;
char * path, argumentv[], envptr[];

Description
The ra_exec subroutine in all its forms, executes a new program in the calling process, and attaches the process to the resource specified by the rstype and rsid parameters.

The following conditions must be met to successfully attach a process to a resource set:
- The resource set must contain processors that are available in the system.
- The process must either have root authority or have CAP_NUMA_ATTACH capability.
- The calling thread must not have a bindprocessor binding to a processor.
- The resource set must be contained in (be a subset of ) the process' partition resource set.

Note: When the exec subroutine is used, the new process image inherits its process' resource set attachments.

Dynamic Processor Deallocation and DLPAR may invalidate the processor attachment that is being specified. A program must become DLPAR Aware to resolve this problem.

The flags parameter can be set to indicate the policy for using the resources contained in the resource set specified in the rset parameter. The only supported policy today is RS_STRSET, which is useful only when the processors of the system are running in simultaneous multi-threading mode. Processors like the POWER5 support simultaneous multi-threading, where each physical processor has two execution engines, called hardware threads. Each hardware thread is essentially equivalent to a single CPU, and each is identified as a separate CPU in a resource set. The RS_STRSET flag indicates that the process is to be scheduled with a single-threaded policy; namely, that it should be scheduled on only one hardware thread per physical processor. All the available processors indicated in the resource set are required to specify this flag for exclusive use. A new resource set, called an ST resource set, is constructed from the specified resource set and attached to the process according to the following rules:
- All offline processors are ignored.
- If all the hardware threads (CPUs) of a physical processor (when running in simultaneous multi-threading mode, there will be more than one active hardware thread per physical processor) are not included in the specified resource set, the other CPUs of the processor are ignored when constructing the ST resource set.
- Only one CPU (hardware thread) resource per physical processor is included in the ST resource set.

Parameters
The ra_exec subroutine has the same parameters as the exec subroutine, with the addition of the following new parameters:

rstype Specifies the type of resource the new process image will be attached to. This parameter must be the following, defined in rset.h:
- R_RSET: resource set

rsid Identifies the resource the new process image will be attached to. This parameter must be a resource set handle.
- Process ID (for rstype of R_PROCESS): set the rsid_t at_pid field to the desired process' process ID.

flags Specifies the policy to use for the rset. A value of RS_STRSET indicates that the process is to be scheduled with a single-threaded policy (only on one hardware thread per physical processor).
Return Values
The ra_exec subroutine’s return values are the same as the exec subroutine’s return values.

Error Codes
The ra_exec subroutine’s error codes are the same as the exec subroutine’s error codes, with the addition of the following error codes:

EINVAL  One of the following is true:
    • The rstype parameter contains an invalid type identifier.
    • The flags parameter contains an invalid flags value.
    • The RS_STRSET flags parameter is specified and one or more processors in the rset parameter are not assigned for exclusive use.

ENODEV  The resource set specified by the rset parameter does not contain any available processors, or the RS_STRSET flags parameter is specified and the constructed ST resource set does not have any available processors.

EFAULT  Invalid address.

EPERM  One of the following is true:
    • The calling process has neither root authority nor CAP_NUMA_ATTACH attachment privilege.
    • The calling process contains one or more threads with a bindprocessor processor binding.
    • The specified resource set is not included in the calling process’ partition resource set.

Related Information
The "ra_fork Subroutine," "ra_attachrset Subroutine" on page 10, "ra_detachrset Subroutine" on page 12, and "ra_getrset Subroutine" on page 21.

The Dynamic Logical Partitioning article in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

The dr_reconfig system call in AIX 5L Version 5.3 Technical Reference: Kernel and Subsystems Volume 1.


ra_fork Subroutine

Purpose
Creates and attaches a new process to a given resource.

Library
Standard C library (libc.a)

Syntax
#include <sys/rset.h>

pid_t ra_fork(rstype, rsid, flags);

Description
The ra_fork subroutine creates a new process, and attaches the new process to the resource set specified by rstype and rsid.
The following conditions must be met to successfully attach a process to a resource set:

- The resource set must contain processors that are available in the system.
- The process must either have root authority or have CAP_NUMA_ATTACH capability.
- The calling thread must not have a bindprocessor binding to a processor.
- The resource set must be contained in (be a subset of) the process’ partition resource set.

**Note:** When the `fork` subroutine is used, the child process inherits its parent’s resource set attachments.

Dynamic Processor Deallocation and DLPAR may invalidate the processor attachment that is being specified. A program must become DLPAR Aware to resolve this problem.

The `flags` parameter can be set to indicate the policy for using the resources contained in the resource set specified in the `rset` parameter. The only supported policy today is RS_STRSET, which is useful only when the processors of the system are running in simultaneous multi-threading mode. Processors like the POWER5 support simultaneous multi-threading, where each physical processor has two execution engines, called hardware threads. Each hardware thread is essentially equivalent to a single CPU, and each is identified as a separate CPU in a resource set. The RS_STRSET flag indicates that the process is to be scheduled with a single-threaded policy; namely, that it should be scheduled on only one hardware thread per physical processor. All the available processors indicated in the resource set are required to specify this flag for exclusive use. A new resource set, called an ST resource set, is constructed from the specified resource set and attached to the process according to the following rules:

- All offline processors are ignored.
- If all the hardware threads (CPUs) of a physical processor (when running in simultaneous multi-threading mode, there will be more than one active hardware thread per physical processor) are not included in the specified resource set, the other CPUs of the processor are ignored when constructing the ST resource set.
- Only one CPU (hardware thread) resource per physical processor is included in the ST resource set.

**Parameters**

- `rstype`: Specifies the type of resource the new process will be attached to. This parameter must be the following value, defined in `rset.h`.
  - R_RSET: resource set.
- `rsid`: Identifies the resource the new process will be attached to. This parameter must be a resource set handle.
  - Resource set ID (for `rstype` of R_RSET): set the `rsid_t at rset` field to the desired resource set.
- `flags`: Specifies the policy to use for the `rset`. A value of RS_STRSET indicates that the process is to be scheduled with a single-threaded policy (only on one hardware thread per physical processor).

**Return Values**

The `ra_fork` subroutine’s return values are the same as the `fork` subroutine’s return values.

**Error Codes**

The `ra_fork` subroutine’s error codes are the same as the `fork` subroutine’s error codes with the addition of the following:

- `EINVAL`: One of the following is true:
  - The `rstype` parameter contains an invalid type identifier.
  - The `flags` parameter contains an invalid flags value.
  - The RS_STRSET `flags` parameter is specified and one or more processors in the `rset` parameter are not assigned for exclusive use.
ENODEV  The resource set specified by the rset parameter does not contain any available processors, or the RS_STRSET flags parameter is specified and the constructed ST resource set does not have any available processors.

EFAULT  Invalid address.

EPERM  One of the following is true:
   - The calling process has neither root authority nor CAP_NUMA_Attach attachment privilege.
   - The calling process contains one or more threads with a bindprocessor processor binding.
   - The specified resource set is not included in the calling process' partition resource set.

Related Information
"ra_attachrset Subroutine" on page 10, "ra_detachrset Subroutine" on page 12, and "ra_getrset Subroutine" on page 21.

The Dynamic Logical Partitioning article in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

The dr_reconfig system call in AIX 5L Version 5.3 Technical Reference: Kernel and Subsystems Volume 1.

The fork, f_fork, or vfork Subroutine and exec: execl, execle, execlp, execv, execve, execvp, or exect Subroutine articles in AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 2.

ra_free_attachinfo Subroutine

Purpose
Frees the memory allocated for the attachment information returned by ra_get_attachinfo.

Library
Standard C library (libc.a)

Syntax
#include <sys/rset.h>

int ra_free_attachinfo_t(attachinfo_t *info);

Description
The ra_free_attachinfo subroutine frees the memory allocated by ra_get_attachinfo to contain the attachinfo_t structures returning the attachment information.

Parameters
info Pointer to the attachinfo_t structure that was returned by a previous call to ra_get_attachinfo.

Return Values
On successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the errno global variable is set to indicate the error.
Error Codes
The `ra_free_attachinfo` subroutine is unsuccessful if the following is true:

EINVAL
The `info` parameter is a null pointer.

Related Information
The `ra_get_attachinfo Subroutine`.

ra_get_attachinfo Subroutine

Purpose
Retrieves the resource set attachments to which a work component is attached.

Library
Standard C library (`libc.a`)

Syntax
```
#include <sys/rset.h>
attachinfo_t *ra_get_attachinfo(rstype rstype, rsid rsid, offset offset, length length, flags flags);
```

Description
The `ra_get_attachinfo` subroutine retrieves information describing the attachments involving the work component specified by `rstype` and `rsid`.

This information is returned as a null-terminated linked list of `attachinfo_t` structures. The `attachinfo_t` structures are allocated in the caller's process heap. The `ra_free_attachinfo` subroutine is provided to free the list of `attachinfo_t` structures returned by `ra_get_attachinfo`.

The `ra_get_attachinfo` subroutine retrieves attachment information for the following work components:
- A shared memory object identified by a shared memory segment ID.
- A file identified by an open file descriptor.
- An address range in one of the above work components identified by its `offset` in the object and its `length`.

If `rstype` is a memory object and `length` has a 0 value, the attachment information returned is for the last portion of the memory object, beginning with `offset`.

Note: Resource set attachments can change during or after `ra_get_attachinfo` retrieves them. There is no guarantee that the returned attachments still exist, or that all existing attachments were retrieved.
Parameters

rstype
Specifies the type of work component for which the attachment information is to be retrieved. This parameter can have one of the following values:

- **R_SHM**
  Attachment information of a shared memory, identified by its shared memory identifier, is to be retrieved.

- **R_FILDES**
  Attachment information of a file, identified by its open file descriptor, is to be retrieved.

rsid
Identifies the work component for which the attachment information is to be retrieved. This parameter can be one of the following:

- shared memory segment ID (if the value of rstype is R_SHM)
- open file descriptor (if the value of rstype is R_FILDES)

offset
Specifies the offset of a range within a memory object for which the attachment information is to be retrieved. This parameter is taken into account only for the following values of rstype:

- R_SHM: starting offset within the shared memory object identified by rsid
- R_FILDES: absolute offset within the file identified by rsid

length
Specifies the length of a range within a memory object for which the attachment information is to be retrieved. This parameter is taken into account only for the following values of rstype:

- R_SHM: length of a range within the shared memory object identified by rsid
- R_FILDES: length of a range within the file identified by rsid

flags
Reserved for future use. Specify as 0.

Return Values
On successful completion, a pointer to the first element in a null-terminated list of attachinfo_t structures is returned. A null pointer is returned if the work component does not have any attachments. Otherwise, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes
The ra_get_attachinfo subroutine is unsuccessful if one or more of the following are true:

- **EINVAL**
  One of the following conditions is true:
  - The flags parameter contains an invalid value.
  - The rstype parameter contains an invalid type qualifier.
  - The rstype parameter is R_SHM and rsid is not a valid shared memory segment.

- **EBADF**
  The rstype parameter is R_FILDES and rsid is not a valid open file descriptor.
ra_getrset Subroutine

Purpose
Gets the resource set to which a work component is attached.

Library
Standard C library (libc.a)

Syntax
```c
#include <sys/rset.h>
int ra_getrset (rtype_t rstype, rsid_t rsid, flags_t flags, rsethandle_t rset);
```

Description
The `ra_getrset` subroutine returns the resource set to which a specified work component is attached.

The work component is an existing process identified by the process ID or an existing kernel thread identified by the kernel thread ID (tid). A process ID or thread ID value of RS_MYSELF indicates the resource set attached to the current process or the current kernel thread, respectively, is requested.

The following return values from the `ra_getrset` subroutine indicate the type of resource set returned:
- A value of RS_EFFECTIVE_RSET indicates the process was explicitly attached to the resource set. This may have been done with the `ra_attachrset` subroutine.
- A value of RS_PARTITION_RSET indicates the process was not explicitly attached to a resource set. However, the process had an explicitly set partition resource set. This may be set with the `rs_setpartition` subroutine or through the use of WLM work classes with resource sets.
- A value of RS_DEFAULT_RSET indicates the process was not explicitly attached to a resource set nor did it have an explicitly set partition resource set. The system default resource set is returned.
- A value of RS_THREAD_RSET indicates the kernel thread was explicitly attached to the resource set. This might have been done with the `ra_attachrset` subroutine.

Parameters

- **rtype**
  Specifies the type of the work component whose resource set attachment is requested. This parameter must be the following value, defined in `rset.h`:
  - R_PROCESS: existing process
  - R_THREAD: existing kernel thread

- **rsid**
  Identifies the work component whose resource set attachment is requested. This parameter must be the following:
  - Process ID (for `rtype` of R_PROCESS): set the `rsid_t at_pid` field to the desired process’ process ID.
  - Kernel thread ID (for `rtype` of R_THREAD): set the `rsid_t at_tid` field to the desired kernel thread’s thread ID.

- **flags**
  Reserved for future use. Specify as 0.

- **rset**
  Specifies the resource set to receive the work component’s resource set.
Return Values
If successful, a value of RS_EFFECTIVE_RSET, RS_PARTITION_RSET, RS_THREAD_RSET, or RS_DEFAULT_RSET is returned. If unsuccessful, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes
The ra_getrset subroutine is unsuccessful if one or more of the following are true:

EINVAL One of the following is true:
- The flags parameter contains an invalid value.
- The rstype parameter contains an invalid type qualifier.

EFAULT Invalid address.

ESRCH The process or kernel thread identified by the rstype and rsid parameters does not exist.

Related Information
The "rs_getpartition Subroutine" on page 109.

ra_mmap or ra_mmapv Subroutine

Purpose
Maps a file or anonymous memory region into the process-address space and attaches the file or memory region to a given resource.

Library
Standard C Library (libc.a)

Syntax
#include <rset.h>
#include <sys/mman.h>

void *ra_mmap(addr, len, prot, flags, fildes, off, rstype, rsid, att_flags);

void *ra_mmapv(addr, len, prot, flags, fildes, off, rangecnt, rangevec);

Description
The ra_mmap subroutine maps the file or memory region, specified by mmap_params, into the process-address space and attaches it to the resource set specified by rstype and rsid. The resource set specified for attachment defines the resource allocation domains (RADs) from which the mapping's
memory demands should be fulfilled. If the file or memory region is attached to a resource set specifying multiple RADs, its memory allocation is distributed among these RADs according to att_flags.

If a file is being mapped, the attachment for the new mapped region is reflected down to the portion of the file it maps and persists after the region is unmapped. The file’s attachment persists until the last close of the file.

The ra_mmapv subroutine is similar to the ra_mmap subroutine, and allows multiple subranges of a file or memory region to be attached to different resource sets in a single ra_mmapv call.

The rangelcnt argument specifies the number of subranges being mapped. The rangevec argument is a pointer to an array of subrange_t structures describing the attachments to be performed. Each subrange_t structure specifies a portion of the file or memory region and the resource set to which the portion should be attached. If overlapping subranges are specified, ra_mmapv does not fail, but its behavior is undefined.

Child processes inherit all mapped regions and their resource set attachments from the parent process when the fork subroutine is called. The child process also inherits the same sharing and protection attributes for these mapped regions. A successful call to any exec subroutine unmaps all mapped regions created with the ra_mmap subroutine.

Attachments to a given RAD do not attach the process to the processors in that RAD. Attachments are only advisory; memory from a different RAD can be provided if the demand cannot be fulfilled from the RAD specified.

If overlapping subranges are mapped with attachments, the memory placement of the mapped regions is undefined.

The su_rsoffset and su_rslength fields of the subrange_t structures must be set to 0. Otherwise, ra_mmapv fails with EINVAL.

Parameters

addr
Specifies the starting address of the memory region to be mapped. When the MAP_FIXED flag is specified, this address must be a multiple of the page size returned by the sysconf subroutine using the _SC_PAGE_SIZE value for the Name parameter. A region is never placed at address 0, or at an address where it would overlap an existing region.

att_flags
Specifies how memory allocation is distributed among the RADs.

fildes
Specifies the file descriptor of the file-system object or of the shared memory object to be mapped. If the MAP_ANONYMOUS flag is set, the fildes parameter must be -1. After the successful completion of the ra_mmap or ra_mmapv subroutine, the file or the shared memory object specified by the fildes parameter can be closed without affecting the mapped region or the contents of the mapped file. Each mapped region creates a file reference, similar to an open file descriptor, which prevents the file data from being deallocated.
flags

Specifies attributes of the mapped region. Values for the flags parameter are constructed by a bitwise-inclusive ORing of values from the following list of symbolic names defined in the sys/mman.h file:

**MAP_FILE**

Specifies the creation of a new mapped file region by mapping the file associated with the fildes file descriptor. The mapped region can extend beyond the end of the file, both at the time when the ra_mmap subroutine is called and while the mapping persists. This situation could occur if a file with no contents was created just before the call to the ra_mmap subroutine, or if a file was later truncated. However, references to whole pages following the end of the file result in the delivery of a SIGBUS signal. Only one of the MAP_FILE and MAP_ANONYMOUS flags must be specified with the ra_mmap or ra_mmapv subroutine.

**MAP_ANONYMOUS**

Specifies the creation of a new, anonymous memory region that is initialized to all zeros. This memory region can be shared only with the descendants of the current process. When using this flag, the fildes parameter must be -1. Only one of the MAP_FILE and MAP_ANONYMOUS flags must be specified with the ra_mmap or ra_mmapv subroutine.

**MAP_VARIABLE**

Specifies that the system select an address for the new memory region if the new memory region cannot be mapped at the address specified by the addr parameter, or if the addr parameter is null. Only one of the MAP_VARIABLE and MAP_FIXED flags must be specified with the ra_mmap or ra_mmapv subroutine.

**MAP_FIXED**

Specifies that the mapped region be placed exactly at the address specified by the addr parameter. If the application has requested SPEC1170 complaint behavior and the ra_mmap or ra_mmapv request is successful, the mapping replaces any previous mappings for the process’ pages in the specified range. If the application has not requested SPEC1170 compliant behavior and a previous mapping exists in the range, the request fails. Only one of the MAP_VARIABLE and MAP_FIXED flags must be specified with the ra_mmap or ra_mmapv subroutine.

**MAP_SHARED**

When the MAP_SHARED flag is set, modifications to the mapped memory region will be visible to other processes that have mapped the same region using this flag. If the region is a mapped file region, modifications to the region will be written to the file.

You can specify only one of the MAP_SHARED or MAP_PRIVATE flags with the ra_mmap or ra_mmapv subroutine. MAP_PRIVATE is the default setting when neither flag is specified unless you request SPEC1170 compliant behavior. In this case, you must choose either MAP_SHARED or MAP_PRIVATE.

**MAP_PRIVATE**

When the MAP_PRIVATE flag is specified, modifications to the mapped region by the calling process are not visible to other processes that have mapped the same region. If the region is a mapped file region, modifications to the region are not written to the file.

If this flag is specified, the initial write reference to an object page creates a private copy of that page and redirects the mapping to the copy. Until then, modifications to the page by processes that have mapped the same region with the MAP_SHARED flag are visible.

You can specify only one of the MAP_SHARED or MAP_PRIVATE flags with the ra_mmap or ra_mmapv subroutine. MAP_PRIVATE is the default setting when neither flag is specified unless you request SPEC1170 compliant behavior. In this case, you must choose either MAP_SHARED or MAP_PRIVATE.

len

Specifies the length, in bytes, of the memory region to be mapped. The system performs mapping operations over whole pages only. If the len parameter is not a multiple of the page size, the system will include in any mapping operation the address range between the end of the region and the end of the page containing the end of the region.
Specifies the file byte offset at which the mapping starts. This offset must be a multiple of the page size returned by the `sysconf` subroutine using the `_SC_PAGE_SIZE` value for the `Name` parameter.

Specifies an advisory memory allocation policy that is to be applied. This parameter must have one of the following values defined in `rset.h`:

- **P_FIRST_TOUCH**
  First Access memory policy. Memory is allocated from the RAD of the processor on which it is accessed the first time if this RAD is in the attachment resource set. Otherwise, memory is allocated from any RAD with memory available to the processor.

- **P_BALANCED**
  Balanced memory policy. Memory is allocated in a round robin manner across the RADs contained in the attachment resource set.

- **P_DEFAULT**
  Default memory placement policy.

Specifies the access permissions for the mapped region. The `sys/mman.h` file defines the following access options:

- **PROT_READ**
  Region can be read.

- **PROT_WRITE**
  Region can be written.

- **PROT_EXEC**
  Region can be executed.

- **PROT_NONE**
  Region cannot be accessed.

The `prot` parameter can be the `PROT_NONE` flag, or any combination of the `PROT_READ` flag, `PROT_WRITE` flag, and `PROT_EXEC` flag logically ORed together. If the `PROT_NONE` flag is not specified, access permissions can be granted to the region in addition to those explicitly requested. However, write access will not be granted unless the `PROT_WRITE` flag is specified.

Note: The operating system generates a `SIGSEGV` signal if a program attempts an access that exceeds the access permission given to a memory region. For example, if the `PROT_WRITE` flag is not specified and a program attempts a write access, a `SIGSEGV` signal results. If the region is a mapped file that was mapped with the `MAP_SHARED` flag, the `ra_mmap` or `ra_mmapv` subroutine grants read or execute access permission only if the file descriptor used to map the file was opened for reading. It grants write access permission only if the file descriptor was opened for writing. If the region is a mapped file that was mapped with the `MAP_PRIVATE` flag, the `ra_mmap` or `ra_mmapv` subroutine grants read, write, or execute access permission only if the file descriptor used to map the file was opened for reading. If the region is an anonymous memory region, the `ra_mmap` or `ra_mmapv` subroutine grants all requested access permissions.

Specifies the number of `subrange_t` structures pointed to by `rangevec`.

Specifies a pointer to an array of `subrange_t` structures describing the desired subrange attachments.

Identifies the resource set to which the file or memory region is to be attached. This structure must contain a resource set handle of an existing resource set. Attachments to resources are advisory. If memory cannot be allocated from the RADs contained in the attachment resource set, memory is allocated from any RAD in the system that has memory available.

Specifies the type of resource the file or memory region is to be attached to. This parameter must have the resource set value `R_RSET`, defined in `rset.h`.
Return Values

Upon successful completion, an address to the mapped file or memory region is returned. Otherwise, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes

**EACCES**  The file referred to by the fildes parameter is not open for read access, or the file is not open for write access and the PROT_WRITE flag was specified for a MAP_SHARED mapping operation. Or, the file to be mapped has enforced locking enabled and the file is currently locked.

**EAGAIN**  The fildes parameter refers to a device that has already been mapped.

**EBADF**  The fildes parameter is not a valid file descriptor, or the MAP_ANONYMOUS flag was set and the fildes parameter is not -1.

**EFBIG**  The mapping requested extends beyond the maximum file size associated with fildes.

**EINVAL**  The flags or prot parameter is invalid, or the addr parameter or off parameter is not a multiple of the page size returned by the sysconf subroutine using the _SC_PAGE_SIZE value for the Name parameter.

**EINVAL**  The application has requested SPEC1170 compliant behavior and the value of flags is invalid (neither MAP_PRIVATE nor MAP_SHARED is set).

**EINVAL**  The subrange_t structure specifies an invalid range.

**EINVAL**  The su_rsoffset and su_rslength fields of a subrange_t do not have a value of 0.

**EINVAL**  The resource type is invalid (is not of type R_RSET).

**EINVAL**  The application has requested SPEC1170 compliant behavior and the number of mapped regions would exceed an implementation-dependent limit (per process or per system).

**ENOMEM**  There is not enough address space to map len bytes, or the application has not requested Single UNIX Specification, Version 2 compliant behavior and the MAP_FIXED flag was set and part of the address-space range (addr, addr+len) is already allocated.

**ENODEV**  The fildes parameter refers to an object that cannot be mapped, such as a terminal.

**ENOMEM**  There is not enough address space to map len bytes, or the application has not requested Single UNIX Specification, Version 2 compliant behavior and the MAP_FIXED flag was set and part of the address-space range (addr, addr+len) is already allocated.

**ENOSYS**  The ra_mmap subroutine is not supported on the system.

**ENOSYS**  The file specified is of a type that does not support physical attachments.

**ENXIO**  The addresses specified by the range (off, off+len) are invalid for the fildes parameter.

**EOVERFLOW**  The mapping requested extends beyond the offset maximum for the file description associated with fildes.

**EPERM**  The calling process does not have the necessary attachment privileges.

Related Information

The** mmap Subroutine**, **ra_attachrset Subroutine** on page 10, **ra_detachrset Subroutine** on page 12, **ra_exec Subroutine** on page 14, **ra_fork Subroutine** on page 16, **ra_shmget and ra_shmgetv Subroutines**, **rs_alloc Subroutine** on page 101, **rs_free Subroutine** on page 103, **rs_getassociativity Subroutine** on page 104, **rs_getinfo Subroutine** on page 105, **rs_getrad Subroutine** on page 110.

The** mkrset Command**

ra_shmget and ra_shmgetv Subroutines

**Purpose**

Gets a shared memory segment and attaches it to a resource.

**Library**

Standard C library (libc.a)
## Syntax

```c
#include <rset.h>
#include <sys/shm.h>

int ra_shmget(key, size, flags, rstype, rsid, att_flags);
key_t key;
size64_t size;
int flags;
rstype_t rstype;
rsid_t rsid;
unsigned int att_flags;
int ra_shmgetv(key, size, flags, rangecnt, rangevec);
key_t key;
size64_t size;
int flags;
rangecnt;
subrange_t *rangevec;
```

## Parameters

As per existing `shmget` usage, plus the following new parameters:

- **rstype**
  Specifies the type of resource the new shared memory segment is to be attached to. This parameter must have the resource set value `R_RSET`, defined in `rset.h`.

- **rsid**
  Identifies the resource to which the new shared memory segment is to be attached. This parameter must be a resource set handle of an existing resource set. If memory cannot be allocated from the RADs contained in the attachment resource set, memory is allocated from any RAD in the system that has memory available.

- **att_flags**
  Specifies an advisory memory allocation policy that is to be applied to the new shared memory segment. This parameter must have one of the following values defined in `rset.h`:

  - **P_FIRST_TOUCH**: First Access memory policy. Memory is allocated from the current node, the RAD of the processor on which it is accessed for the first time, if this RAD is in the attachment resource set. If it is not, memory is allocated form an undefined RAD in the attachment resource set.
  
  - **P_BALANCED**: Balanced memory policy. Memory is allocated in a round robin manner across the RADs contained in the attachment resource set.
  
  - **P_DEFAULT**: Default memory placement policy.

- **rangecnt**
  Specifies the number of `subrange_t` structures pointed to by `rangevec`.

- **rangevec**
  Specifies a pointer to an array of `subrange_t` structures describing the desired subrange attachments.

## Description

The `ra_shmget` subroutine returns the shared memory identifier associated with the specified `key`, `size` and `flags` parameters, attaching it to the logical or physical resource set (`R_RSET`) specified by `rstype` and `rsid`. If the shared memory is attached to a set of physical resources involving multiple resource allocation domains (RADs), its memory allocation is distributed among these RADs according to `att_flags`. The processors listed in a resource set are used for memory associativity; `rset` memory regions are ignored. Any memory allocation policy is advisory.

If the new shared memory segment is to be attached in its entirety to a resource (that is, no subranges are involved), then the `rstype` parameter is of type `R_RSET` and the `rsid` parameter is a resource set handle.

The `ra_shmgetv` subroutine is similar to the `ra_shmget` subroutine, and allows multiple subranges of the new shared memory segment to be attached to multiple resources in a single `ra_shmgetv` call. The `rangevec` argument is a pointer to an array of `subrange_t` structures describing the attachments to be performed. The `rangecnt` argument specifies the number of `subrange_t` structures pointed to by `rangevec`. All unused `subrange_t` structure fields, including those marked as reserved, must be initialized to the value of 0. Although it is not failing, the behavior with overlapping subranges is undefined.
Return Values
On successful completion, a shared memory identifier is returned. Otherwise, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes
As per existing shmget usage, plus the following errors:

EINVAL
One of the following conditions is true:
- rstype contains an invalid type qualifier.
- Invalid subrange fields.
- att_flags contains an invalid flag.

EPERM
One of the following conditions is true:
- The calling process has neither root authority nor CAP_NUMA_ATTACH privilege.
- The resource specified by rstype and rsid is not included in the calling process's partition resource set.

Examples
The following example attempts to use ra_shmgetv to create a shmat attachable shared memory region, whose first 32 megabytes are distributed using the P_BALANCED policy and the next 48 megabytes using the P_FIRST_TOUCH policy.

```c
int flags, shm_id;
char *shm_at;
rsethandle_t rsetid;
subrange_t subranges[2] = { 0 };

rsetid = rs_alloc(RS_PARTITION);
subranges[0].su_offset = 0x00000000;
subranges[0].su_length = 0x2000000;
subranges[0].su_rstype = R_RSET;
subranges[0].su_rsid.at_rset = rsetid;
subranges[0].su_policy = P_BALANCED;
subranges[1].su_offset = 0x2000000;
subranges[1].su_length = 0x3000000;
subranges[1].su_rstype = R_RSET;
subranges[1].su_rsid.at_rset = rsetid;
subranges[1].su_policy = P_FIRST_TOUCH;

flags = (IPC_CREAT | SHM_PIN);
shm_id = ra_shmgetv (IPC_PRIVATE, 0x5000000, flags,
sizeof(subranges) / sizeof(subrange_t), subranges);
if (shm_id == -1)
{
    perror("ra_shmgetv failed!\n");
    exit(1);
}
```

Implementation Specifics
The ra_shmget and ra_shmgetv subroutines are part of the Base Operating System (BOS) Runtime.

Related Information
The "raAttachrset Subroutine" on page 10, "raDetachrset Subroutine" on page 12.
read, readx, readv, readvx, or pread Subroutine

Purpose
Reads from a file.

Library
Standard C Library (libc.a)

Syntax
```c
#include <unistd.h>

ssize_t read (FileDescriptor, Buffer, NBytes)
int FileDescriptor;
void * Buffer;
size_t NBytes;

int readx (FileDescriptor, Buffer, NBytes, Extension)
int FileDescriptor;
char * Buffer;
unsigned int NBytes;
int Extension;
#include <sys/uio.h>

ssize_t readv (FileDescriptor, iov, iovCount)
int FileDescriptor;
const struct iovec * iov;
int iovCount;

ssize_t readvx (FileDescriptor, iov, iovCount, Extension)
int FileDescriptor;
struct iovec * iov;
int iovCount;
int Extension;
#include <unistd.h>

ssize_t pread (int fildes, void *buf, size_t nbyte, off_t offset);
```

Description
The `read` subroutine attempts to read \( NBytes \) of data from the file associated with the `FileDescriptor` parameter into the buffer pointed to by the `Buffer` parameter.

The `readv` subroutine performs the same action but scatters the input data into the `iovCount` buffers specified by the array of `iovec` structures pointed to by the `iov` parameter. Each `iovec` entry specifies the base address and length of an area in memory where data should be placed. The `readv` subroutine always fills an area completely before proceeding to the next.

The `readx` and `readvx` subroutines are the same as the `read` and `readv` subroutines, respectively, with the addition of an `Extension` parameter, which is needed when reading from some device drivers and
when reading directories. While directories can be read directly, it is recommended that the `opendir` and `readdir` calls be used instead, as this is a more portable interface.

On regular files and devices capable of seeking, the `read` starts at a position in the file given by the file pointer associated with the `FileDescriptor` parameter. Upon return from the `read` subroutine, the file pointer is incremented by the number of bytes actually read.

Devices that are incapable of seeking always read from the current position. The value of a file pointer associated with such a file is undefined.

On directories, the `readvx` subroutine starts at the position specified by the file pointer associated with the `FileDescriptor` parameter. The value of this file pointer must be either 0 or a value which the file pointer had immediately after a previous call to the `readvx` subroutine on this directory. Upon return from the `readvx` subroutine, the file pointer increments by a number that may not correspond to the number of bytes copied into the buffers.

When attempting to read from an empty pipe (first-in-first-out (FIFO)):
- If no process has the pipe open for writing, the `read` returns 0 to indicate end-of-file.
- If some process has the pipe open for writing:
  - If `O_NDELAY` and `O_NONBLOCK` are clear (the default), the `read` blocks until some data is written or the pipe is closed by all processes that had opened the pipe for writing.
  - If `O_NDELAY` is set, the `read` subroutine returns a value of 0.
  - If `O_NONBLOCK` is set, the `read` subroutine returns a value of -1 and sets the global variable `errno` to `EAGAIN`.

When attempting to read from a character special file that supports nonblocking reads, such as a terminal, and no data is currently available:
- If `O_NDELAY` and `O_NONBLOCK` are clear (the default), the `read` subroutine blocks until data becomes available.
- If `O_NDELAY` is set, the `read` subroutine returns 0.
- If `O_NONBLOCK` is set, the `read` subroutine returns -1 and sets the `errno` global variable to `EAGAIN` if no data is available.

When attempting to read a regular file that supports enforcement mode record locks, and all or part of the region to be read is currently locked by another process:
- If `O_NDELAY` and `O_NONBLOCK` are clear, the `read` blocks the calling process until the lock is released.
- If `O_NDELAY` or `O_NONBLOCK` is set, the `read` returns -1 and sets the global variable `errno` to `EAGAIN`.

The behavior of an interrupted `read` subroutine depends on how the handler for the arriving signal was installed.

If the handler was installed with an indication that subroutines should not be restarted, the `read` subroutine returns a value of -1 and the global variable `errno` is set to `EINTR` (even if some data was already consumed).

If the handler was installed with an indication that subroutines should be restarted:
- If no data had been read when the interrupt was handled, this `read` will not return a value (it is restarted).
- If data had been read when the interrupt was handled, this `read` subroutine returns the amount of data consumed.
The `pread` function performs the same action as `read`, except that it reads from a given position in the file without changing the file pointer. The first three arguments to `pread` are the same as `read` with the addition of a fourth argument offset for the desired position inside the file. An attempt to perform a `pread` on a file that is incapable of seeking results in an error.

**Note:** The `pread64` subroutine applies to AIX 4.3 and later.

```c
ssize_t pread64(int fildes, void *buf, size_t nbytes, off64_t offset)
```

The `pread64` subroutines performs the same action as `pread` but the limit of offset to the maximum file size for the file associated with the file Descriptor and DEV_OFF_MAX if the file associated with fileDescriptor is a block special or character special file. If `fildes` refers to a socket, `read` is equivalent to the `recv` subroutine with no flags set.

Using the `read` or `pread` subroutine with a file descriptor obtained from a call to the `shm_open` subroutine fails with `ENXIO`.

**Parameters**

**FileDescriptor**

A file descriptor identifying the object to be read.

**Extension**

Provides communication with character device drivers that require additional information or return additional status. Each driver interprets the `Extension` parameter in a device-dependent way, either as a value or as a pointer to a communication area. Drivers must apply reasonable defaults when the value of the `Extension` parameter is 0.

For directories, the `Extension` parameter determines the format in which directory entries should be returned:

- If the value of the `Extension` parameter is 0, the format in which directory entries are returned depends on the value of the `real directory read` flag (described in the `ulimit` subroutine).
- If the calling process does not have the `real directory read` flag set, the buffers are filled with an array of directory entries truncated to fit the format of the System V directory structure. This provides compatibility with programs written for UNIX System V.
- If the calling process has the `real directory read` flag set (see the `ulimit` subroutine), the buffers are filled with an image of the underlying implementation of the directory.
- If the value of the `Extension` parameter is 1, the buffers are filled with consecutive directory entries in the format of `dirent` structure. This is logically equivalent to the `readdir` subroutine.
- Other values of the `Extension` parameter are reserved.

For tape devices, the `Extension` parameter determines the response of the `readx` subroutine when the tape drive is in variable block mode and the read request is for less than the tape’s block size.

- If the value of the `Extension` parameter is `TAPE_SHORT_READ`, the `readx` subroutine returns the number of bytes requested and sets the `errno` global variable to a value of 0.
- If the value of the `Extension` parameter is 0, the `readx` subroutine returns a value of 0 and sets the `errno` global variable to `ENOMEM`.

**iov**

Points to an array of `iovec` structures that identifies the buffers into which the data is to be placed. The `iovec` structure is defined in the `sys/uio.h` file and contains the following members:

```c
caddr_t iov_base;
size_t iov_len;
```

**iovCount** specifies the number of `iovec` structures pointed to by the `iov` parameter.

**Buffer** Points to the buffer.
NBytes specifies the number of bytes read from the file associated with the FileDescriptor parameter. 

Note: When reading tapes, the read subroutines consume a physical tape block on each call to the subroutine. If the physical data block size is larger than specified by the Nbytes parameter, an error will be returned, since all of the data from the read will not fit into the buffer specified by the read.

To avoid read errors due to unknown blocking sizes on tapes, set the NBytes parameter to a very large value (such as 32K bytes).

Return Values

Upon successful completion, the read, readx, ready, readvx, and pread subroutines return the number of bytes actually read and placed into buffers. The system guarantees to read the number of bytes requested if the descriptor references a normal file that has the same number of bytes left before the end of the file is reached, but in no other case.

A value of 0 is returned when the end of the file has been reached. (For information about communication files, see the ioctl and termio files.)

Otherwise, a value of -1 is returned, the global variable errno is set to identify the error, and the content of the buffer pointed to by the Buffer or iov parameter is indeterminate.

Error Codes

The read, readx, ready, readvx, and pread subroutines are unsuccessful if one or more of the following are true:

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.

EBADF The FileDescriptor parameter is not a valid file descriptor open for reading.

EINVAL The file position pointer associated with the FileDescriptor parameter was negative.

EINVAL The sum of the iov_len values in the iov array was negative or overflowed a 32-bit integer.

EINVAL The value of the iovCount parameter was not between 1 and 16, inclusive.

EINVAL The value of the Nbytes parameter that is larger than OFF_MAX, was requested on the 32-bit kernel. This is a case where the system call is requested from a 64-bit application that is running on a 32-bit kernel.

EINVAL The STREAM or multiplexer referenced by FileDescriptor is linked (directly or indirectly) downstream from a multiplexer.

EAGAIN The file was marked for non-blocking I/O, and no data was ready to be read.

EFAULT The Buffer or part of the iov points to a location outside of the allocated address space of the process.

EDEADLK A deadlock would occur if the calling process were to sleep until the region to be read was unlocked.

EINTR A read was interrupted by a signal before any data arrived, and the signal handler was installed with an indication that subroutines are not to be restarted.

EIO An I/O error occurred while reading from the file system.

EIO The process is a member of a background process attempting to read from its controlling terminal, and either the process is ignoring or blocking the SIGTTIN signal or the process group has no parent process.

EFBIG An offset greater than MAX_FILESIZE was requested on the 32-bit kernel.

ENXIO The read or pread subroutine was used with a file descriptor obtained from a call to the shm_open subroutine.

EOVERFLOW An attempt was made to read from a regular file where NBytes was greater than zero and the starting offset was before the end-of-file and was greater than or equal to the offset maximum established in the open file description associated with FileDescriptor.
The `read`, `readdx`, `readv`, `readdvx` and `pread` subroutines may be unsuccessful if the following is true:

**ENXIO**  A request was made of a nonexistent device, or the request was outside the capabilities of the device.
**ESPIPE**  `fildes` is associated with a pipe or FIFO.

If Network File System (NFS) is installed on the system, the `read` system call can also fail if the following is true:

**ETIMEDOUT**  The connection timed out.

### Related Information

The `fcntl`, `dup`, or `dup2` subroutine, `ioctl` subroutine, `lockfx` subroutine, `lseek` subroutine, `open`, `openx`, or `creat` subroutine, `opendir`, `readdir`, or `seekdir` subroutine, `pipe` subroutine, `poll` subroutine, `socket` subroutine, `socketpair` subroutine.

The `readdir_r` subroutine in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

### readdir_r Subroutine

#### Purpose

Reads a directory.

#### Library

Thread-Safe C Library (libc_r.a)

#### Syntax

```c
#include <sys/types.h>
#include <dirent.h>

int readdir_r (DirectoryPointer, Entry, Result)
DIR * DirectoryPointer;
struct dirent * Entry;
struct dirent ** Result;
```

#### Description

The `readdir_r` subroutine returns the directory entry in the structure pointed to by the `Result` parameter. The `readdir_r` subroutine returns entries for the . (dot) and .. (dot-dot) directories, if present, but never returns an invalid entry (with `d_ino` set to 0). When it reaches the end of the directory, the `readdir_r` subroutine returns 9 and sets the `Result` parameter to NULL. When it detects an invalid `seekdir` operation, the `readdir_r` subroutine returns a 9.

**Note:** The `readdir` subroutine is reentrant when an application program uses different `DirectoryPointer` parameter values (returned from the `opendir` subroutine). Use the `readdir_r` subroutine when multiple threads use the same directory pointer.

Using the `readdir_r` subroutine after the `closedir` subroutine, for the structure pointed to by the `DirectoryPointer` parameter, has an undefined result. The structure pointed to by the `DirectoryPointer` parameter becomes invalid for all threads, including the caller.

Programs using this subroutine must link to the libpthreads.a library.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectoryPointer</td>
<td>Points to the DIR structure of an open directory.</td>
</tr>
<tr>
<td>Entry</td>
<td>Points to a structure that contains the next directory entry.</td>
</tr>
<tr>
<td>Result</td>
<td>Points to the directory entry specified by the Entry parameter.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Indicates that the subroutine was successful.</td>
</tr>
<tr>
<td>9</td>
<td>Indicates that the subroutine was not successful or that the end of the directory was reached. If the user has set the environment variable XPG_SUS_ENV=ON prior to execution of the process, then the SIGXFSZ signal is posted to the process when exceeding the process' file size limit, and the subroutine will always be successful.</td>
</tr>
</tbody>
</table>

Error Codes

If the readdir_r subroutine is unsuccessful, the errno global variable is set to one of the following values:

- **EACCES**: Search permission is denied for any component of the structure pointed to by the DirectoryPointer parameter, or read permission is denied for the structure pointed to by the DirectoryPointer parameter.
- **ENAMETOOLONG**: The length of the DirectoryPointer parameter exceeds the value of the PATH_MAX variable, or a path-name component is longer than the value of NAME_MAX variable while the _POSIX_NO_TRUNC variable is in effect.
- **ENOENT**: The named directory does not exist.
- **ENOTDIR**: A component of the structure pointed to by the DirectoryPointer parameter is not a directory.
- **EMFILE**: Too many file descriptors are currently open for the process.
- **ENFILE**: Too many file descriptors are currently open in the system.
- **EBADF**: The structure pointed to by the DirectoryPointer parameter does not refer to an open directory stream.

Examples

To search a directory for the entry name, enter:

```c
len = strlen(name);
DirectoryPointer = opendir(".");
for (readdir_r(DirectoryPointer, &Entry, &Result); Result != NULL;
     readdir_r(DirectoryPointer, &Entry, &Result))
    if (dp->d_namlen == len && !strcmp(dp->d_name, name)) {
        closedir(DirectoryPointer);
        return FOUND;
    }
closedir(DirectoryPointer);
return NOT_FOUND;
```

Related Information

The close subroutine, exec subroutine, fork subroutine, lseek subroutine, openx open, or creat subroutine, read, readv, readx, or readvx subroutine, scandir or alphasort subroutine.

The opendir, readdir, telldir, seekdir, rewinddir, or closedir subroutine.
readlink Subroutine

Purpose
Reads the contents of a symbolic link.

Library
Standard C Library (libc.a)

Syntax
```c
#include <unistd.h>
int readlink (const char *Path, char *Buffer, size_t BufferSize);
```

Description
The `readlink` subroutine copies the contents of the symbolic link named by the `Path` parameter in the buffer specified in the `Buffer` parameter. The `BufferSize` parameter indicates the size of the buffer in bytes. If the actual length of the symbolic link is less than the number of bytes specified in the `BufferSize` parameter, the string copied into the buffer will be null-terminated. If the actual length of the symbolic link is greater than the number of bytes specified in the `BufferSize` parameter, an error is returned. The length of a symbolic link cannot exceed 1023 characters or the value of the `PATH_MAX` constant. `PATH_MAX` is defined in the `limits.h` file.

Parameters
- `Path` Specifies the path name of the destination file or directory.
- `Buffer` Points to the user’s buffer. The buffer should be at least as large as the `BufferSize` parameter.
- `BufferSize` Indicates the size of the buffer. The contents of the link are null-terminated, provided there is room in the buffer.

Return Values
Upon successful completion, the `readlink` subroutine returns a count of the number of characters placed in the buffer (not including any terminating null character). If the `readlink` subroutine is unsuccessful, the buffer is not modified, a value of -1 is returned, and the `errno` global variable is set to indicate the error.

Error Codes
The `readlink` subroutine fails if one or both of the following are true:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENOENT</td>
<td>The file named by the <code>Path</code> parameter does not exist, or the path points to an empty string.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>The file named by the <code>Path</code> parameter is not a symbolic link.</td>
</tr>
<tr>
<td>ERANGE</td>
<td>The path name in the symbolic link is longer than the <code>BufferSize</code> value.</td>
</tr>
</tbody>
</table>

The `readlink` subroutine can also fail due to additional errors. See “Base Operating System Error Codes for Services that Require Path-Name Resolution” for a list of additional error codes.
The `readlink` subroutine can also fail due to additional errors. See Appendix A, “Base Operating System Error Codes for Services That Require Path-Name Resolution” on page A-1 for a list of additional error codes.

If Network File System (NFS) is installed on the system, the `readlink` subroutine can also fail if the following is true:

**ETIMEDOUT** The connection timed out.

---

**Related Information**

The `ln` command.

The `link` subroutine, `statx, stat, fstatx, fstat, fullstat`, or `fullstat`, `fstat64, lstat64, fstat64, fullstat, or lstat64x Subroutine” on page 318 subroutine, `symlink “symlink Subroutine” on page 350` subroutine, `unlink “unlink Subroutine” on page 471` subroutine.

Files, Directories, and File Systems for Programmers in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

**read_real_time or time_base_to_time Subroutine**

**Purpose**

Read the processor real time clock or time base registers to obtain high-resolution elapsed time.

**Library**

Standard C Library (libc.a)

**Syntax**

```c
#include <sys/time.h>
#include <sys/systemcfg.h>

int read_real_time(timebasestruct_t *t, size_t size_of_timebasestruct_t);
int time_base_to_time(timebasestruct_t *t, size_t size_of_timebasestruct_t);
```

**Description**

These subroutines are designed to be used for making high-resolution measurement of elapsed time, using the processor real time clock or time base registers. The `read_real_time` subroutine reads the value of the appropriate registers and stores them in a structure. The `time_base_to_time` subroutine converts time base data to real time, if necessary. This process is divided into two steps because the process of reading the time is usually part of the timed code, and so the conversion from time base to real time can be moved out of the timed code.

The `read_real_time` subroutine reads either the processor real time clock (for the POWER family or PowerPC 601 RISC Microprocessor in AIX 5.1 and earlier) or the time base register (in the case of the POWER-based processors other than the PowerPC 601 RISC Microprocessor). The `t` argument is a pointer to a `timebasestruct_t`, where the time values are recorded.

After calling `read_real_time`, if running on a processor with a real time clock, `t->tb_high` and `t->tb_low` contain the current clock values (seconds and nanoseconds), and `t->flag` contains the `RTC_POWER`. 
If running on a processor with a time base register, \( t->tb\_high \) and \( t->tb\_low \) contain the current values of the time base register, and \( t->flag \) contains \texttt{RTC\_POWER\_PC}.

The \texttt{time\_base\_to\_time} subroutine converts time base information to real time, if necessary. It is recommended that applications unconditionally call the \texttt{time\_base\_to\_time} subroutine rather than performing a check to see if it is necessary.

If \( t->flag \) is \texttt{RTC\_POWER}, the subroutine simply returns (the data is already in real time format).

If \( t->flag \) is \texttt{RTC\_POWER\_PC}, the time base information in \( t->tb\_high \) and \( t->tb\_low \) is converted to seconds and nanoseconds; \( t->tb\_high \) is replaced by the seconds; \( t->tb\_low \) is replaced by the nanoseconds; and \( t->flag \) is changed to \texttt{RTC\_POWER}.

**Parameters**

\( t \)  
Points to a \texttt{timebasestruct\_t}.

**Return Values**

The \texttt{read\_real\_time} subroutine returns \texttt{RTC\_POWER} if the contents of the real time clock has been recorded in the \texttt{timebasestruct}, or returns \texttt{RTC\_POWER\_PC} if the content of the time base registers has been recorded in the \texttt{timebasestruct}.

The \texttt{time\_base\_to\_time} subroutine returns \texttt{0} if the conversion to real time is successful (or not necessary), otherwise \texttt{-1} is returned.

**Examples**

This example shows the time it takes for \texttt{printf} to print the comment between the begin and end time codes:

```c
#include <stdio.h>
#include <sys/time.h>

int main(void)
{
    timebasestruct_t start, finish;
    int val = 3;
    int secs, n_secs;

    /* get the time before the operation begins */
    read_real_time(&start, TIMEBASE_SZ);

    /* begin code to be timed */
    (void) printf("This is a sample line \%d \n", val); /* end code to be timed */

    /* get the time after the operation is complete */
    read_real_time(&finish, TIMEBASE_SZ);

    /*
    * Call the conversion routines unconditionally, to ensure
    * that both values are in seconds and nanoseconds regardless
    * of the hardware platform.
    */
    time_base_to_time(&start, TIMEBASE_SZ);
    time_base_to_time(&finish, TIMEBASE_SZ);

    /* subtract the starting time from the ending time */
    secs = finish.tb_high - start.tb_high;
    n_secs = finish.tb_low - start.tb_low;
```
/*
  * If there was a carry from low-order to high-order during
  * the measurement, we may have to undo it.
  */
if (n_secs < 0) {
  secs--; 
  n_secs += 1000000000;
}

(void) printf("Sample time was %d seconds %d nanoseconds\n", 
              secs, n_secs);

exit(0);
}

Related Information
The \texttt{gettime, settimer, restimer, stime, or time} subroutines, \texttt{getusage, times, or vtimes} subroutines.

High-Resolution Time Measurements Using POWER-based Time Base or POWER family Real-Time Clock in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

\section*{realpath Subroutine}

\subsection*{Purpose}
Resolves path names.

\subsection*{Library}
Standard C Library (libc.a)

\subsection*{Syntax}
\verb+#include <stdlib.h>+  

\verb+char *realpath (const char *file_name,\+  
\verb+char *resolved_name)+

\subsection*{Description}
The \texttt{realpath} subroutine performs filename expansion and path name resolution in \texttt{file_name} and stores it in \texttt{resolved_name}.

The \texttt{realpath} subroutine can handle both relative and absolute path names. For both absolute and relative path names, the \texttt{realpath} subroutine returns the resolved absolute path name.

The character pointed to by \texttt{resolved_name} must be big enough to contain the fully resolved path name. The value of PATH\_MAX (defined in \texttt{limits.h} header file may be used as an appropriate array size.

\subsection*{Return Values}
On successful completion, the \texttt{realpath} subroutine returns a pointer to the resolved name. Otherwise, it returns a null pointer, and sets \texttt{errno} to indicate the error. If the \texttt{realpath} subroutine encounters an error, the contents of \texttt{resolved_name} are undefined.
Error Codes
Under the following conditions, the `realpath` subroutine fails and sets `errno` to:

- **EACCES**
  Read or search permission was denied for a component of the path name.

- **EINVAL**
  File name or resolved name is a null pointer.

- **ELOOP**
  Too many symbolic links are encountered in translating `file_name`.

- **ENAMETOOLONG**
  The length of `file_name` or `resolved_name` exceeds PATH_MAX or a path name component is longer than NAME_MAX.

- **ENOENT**
  The `file_name` parameter does not exist or points to an empty string.

- **ENOTDIR**
  A component of the `file_name` prefix is not a directory.

The `realpath` subroutine may fail if:

- **ENOMEM**
  Insufficient storage space is available.

Related Information
The `getcwd` or `sysconf` subroutine.

reboot Subroutine

**Purpose**
Restarts the system.

**Library**
Standard C Library (`libc.a`)

**Syntax**

```c
#include <sys/reboot.h>

void reboot (int HowTo, void *Argument);

int HowTo;
void *Argument;
```

**Description**

The `reboot` subroutine restarts or re-initial program loads (IPL) the system. The startup is automatic and brings up `/unix` in the normal, nonmaintenance mode.

**Note:** The routine may coredump instead of returning `EFAULT` when an invalid pointer is passed in case of 64-bit application calling 32-bit kernel interface.

The calling process must have root user authority in order to run this subroutine successfully.

**Attention:** Users of the `reboot` subroutine are not portable. The `reboot` subroutine is intended for use only by the `halt`, `reboot`, and `shutdown` commands.
Parameters

**HowTo**
Specifies one of the following values:

- **RB_SOFTIPL**
  Soft IPL.

- **RB_HALT**
  Halt operator; turn the power off.

- **RB_POWIPL**
  Halt operator; turn the power off. Wait a specified length of time, and then turn the power on.

**Argument**
Specifies the amount of time (in seconds) to wait between turning the power off and turning the power on. This option is not supported on all models. Please consult your hardware technical reference for more details.

Return Values

Upon successful completion, the reboot subroutine does not return a value. If the reboot subroutine fails, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes

The reboot subroutine is unsuccessful if any of the following is true:

- **EPERM**
  The calling process does not have root user authority.

- **EINVAL**
  The HowTo value is not valid.

- **EFAULT**
  The Argument value is not a valid address.

Related Information

The halt command, reboot command, shutdown command.

---

**re_comp or re_exec Subroutine**

**Purpose**

Regular expression handler.

**Library**

Standard C Library (libc.a)

**Syntax**

```c
char *re_comp(String)
const char *String;

int re_exec(String)
const char *String;
```

**Description**

**Attention:** Do not use the re_comp or re_exec subroutine in a multithreaded environment.

The re_comp subroutine compiles a string into an internal form suitable for pattern matching. The re_exec subroutine checks the argument string against the last string passed to the re_comp subroutine.
The re_comp subroutine returns 0 if the string pointed to by the String parameter was compiled successfully; otherwise a string containing an error message is returned. If the re_comp subroutine is passed 0 or a null string, it returns without changing the currently compiled regular expression.

The re_exec subroutine returns 1 if the string pointed to by the String parameter matches the last compiled regular expression, 0 if the string pointed to by the String parameter failed to match the last compiled regular expression, and -1 if the compiled regular expression was invalid (indicating an internal error).

The strings passed to both re_comp and re_exec subroutines may have trailing or embedded newline characters; they are terminated by nulls. The regular expressions recognized are described in the manual entry for the ed command, given the above difference.

Parameters

String  Points to a string that is to be matched or compiled.

Return Values

If an error occurs, the re_exec subroutine returns a -1, while the re_comp subroutine returns one of the following strings:

- No previous regular expression
- Regular expression too long
- unmatched \(
- missing ]
- too many \{() pairs
- unmatched \)

Related Information

The compile, step, or advance subroutine, regcmp or regex “regcmp or regex Subroutine” subroutine.

The ed command, sed command, grep command.

List of String Manipulation Services and Subroutines, Example Programs, and Libraries in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.


regcmp or regex Subroutine

Purpose

Compiles and matches regular-expression patterns.

Libraries

Standard C Library ( libc.a )

Programmers Workbench Library (libPW.a)

Syntax

#include <libgen.h>
char *regcmp (String[, String, ...], (char *) 0)
const char *String, ...;

const char *regex (Pattern, Subject[, ret, ...])
char *Pattern, *Subject, *ret, ...;
extern char *__loc1;

Description

Note: The regcmp and regex subroutines are provided for compatibility with existing applications only. For portable applications, use the regcomp and regex subroutine instead.

The regcmp subroutine compiles a regular expression (or Pattern) and returns a pointer to the compiled form. The regcmp subroutine allows multiple String parameters. If more than one String parameter is given, then the regcmp subroutine treats them as if they were concatenated together. It returns a null pointer if it encounters an incorrect parameter.

You can use the regcmp command to compile regular expressions into your C program, frequently eliminating the need to call the regcmp subroutine at run time.

The regex subroutine compares a compiled Pattern to the Subject string. Additional parameters are used to receive values. Upon successful completion, the regex subroutine returns a pointer to the next unmatched character. If the regex subroutine fails, a null pointer is returned. A global character pointer, __loc1, points to where the match began.

The regcmp and regex subroutines are borrowed from the ed command; however, the syntax and semantics have been changed slightly. You can use the following symbols with the regcmp and regex subroutines:

[ ]* . ^
- These symbols have the same meaning as they do in the ed command.
The minus sign (or hyphen) within brackets used with the regex subroutine means “through,” according to the current collating sequence. For example, [a-z] can be equivalent to [abcd . . . xyz] or [aBbCc . . . xYyZz]. You can use the - by itself if the - is the last or first character. For example, the character class expression [ ] -] matches the ] (right bracket) and - (minus) characters.

{ m} {m,} {m, u}
The regcmp subroutine does not use the current collating sequence, and the minus sign in brackets controls only a direct ASCII sequence. For example, [a-z] always means [abc . . . xyz] and [A-Z] always means [ABC . . . XYZ] .
If you need to control the specific characters in a range using the regcmp subroutine, you must list them explicitly rather than using the minus sign in the character class expression.

$ Matches the end of the string. Use the \n character to match a new-line character.
+
A regular expression followed by + (plus sign) means one or more times. For example, [0-9] + is equivalent to [0-9][0-9] *.

{ m} {m,} {m, u}
Integer values enclosed in () (braces) indicate the number of times to apply the preceding regular expression. The m character is the minimum number and the u character is the maximum number. The u character must be less than 256. If you specify only m, it indicates the exact number of times to apply the regular expression. {m,} is equivalent to {m,u} and matches m or more occurrences of the expression. The + (plus sign) and * (asterisk) operations are equivalent to {1,} and {0,}, respectively.

(. . .)$n
This stores the value matched by the enclosed regular expression in the (n+1)th ret parameter. Ten enclosed regular expressions are allowed. The regex subroutine makes the assignments unconditionally.
Parentheses group subexpressions. An operator, such as *, +, or [ ] works on a single character or on a regular expression enclosed in parentheses. For example, (a*(cb+)^)$0.

All of the preceding defined symbols are special. You must precede them with a \ (backslash) if you want to match the special symbol itself. For example, \$ matches a dollar sign.

**Note:** The regcmp subroutine uses the malloc subroutine to make the space for the vector. Always free the vectors that are not required. If you do not free the unneeded vectors, you can run out of memory if the regcmp subroutine is called repeatedly. Use the following as a replacement for the malloc subroutine to reuse the same vector, thus saving time and space:

```c
/* . . . Your Program . . . */
int n;
static int rebuf[256];
return ((n <= sizeof(rebuf)) ? rebuf : NULL);
```

The regcmp subroutine produces code values that the regex subroutine can interpret as the regular expression. For instance, [a-z] indicates a range expression which the regcmp subroutine compiles into a string containing the two end points (a and z).

The regex subroutine interprets the range statement according to the current collating sequence. The expression [a-z] can be equivalent either to [abcd . . . xyz], or to [aBbCcDd . . . xYyZz], as long as the character preceding the minus sign has a lower collating value than the character following the minus sign.

The behavior of a range expression is dependent on the collation sequence. If you want to match a specific set of characters, you should list each one. For example, to select letters a, b, or c, use [abc] rather than [a-c].

**Notes:**
1. No assumptions are made at compile time about the actual characters contained in the range.
2. Do not use multibyte characters.
3. You can use the ] (right bracket) itself within a pair of brackets if it immediately follows the leading [ (left bracket) or [^ (a left bracket followed immediately by a circumflex).
4. You can also use the minus sign (or hyphen) if it is the first or last character in the expression. For example, the expression [ ]-0 matches either the right bracket ( ] ), or the characters - through 0.

**Parameters**

| Subject  | Specifies a comparison string. |
| String   | Specifies the Pattern to be compiled. |
| Pattern  | Specifies the expression to be compared. |
| ret      | Points to an address at which to store comparison data. The regex subroutine allows multiple ret String parameters. |

**Related Information**

The ctype subroutine, compile, step, or advance subroutine, malloc, free, realloc, calloc, mallopt, mallinfo, or alloca subroutine, regcomp ("regcomp Subroutine" on page 44) subroutine, regex ("regexexec Subroutine" on page 47) subroutine.
regcomp Subroutine

Purpose
Compiles a specified basic or extended regular expression into an executable string.

Library
Standard C Library (libc.a)

Syntax
```c
#include <regex.h>
int regcomp (Preg, Pattern, CFlags);
```

Description
The regcomp subroutine compiles the basic or extended regular expression specified by the Pattern parameter and places the output in the structure pointed to by the Preg parameter.

Parameters
- **Preg**: Specifies the structure to receive the compiled output of the regcomp subroutine.
- **Pattern**: Contains the basic or extended regular expression to be compiled by the regcomp subroutine.

The default regular expression type for the Pattern parameter is a basic regular expression. An application can specify extended regular expressions with the REG_EXTENDED flag.
- **CFlags**: Contains the bitwise inclusive OR of 0 or more flags for the regcomp subroutine. These flags are defined in the regex.h file:
  - **REG_EXTENDED**: Uses extended regular expressions.
  - **REG_ICASE**: Ignores case in match.
  - **REG_NOSUB**: Reports only success or failure in the regexec subroutine. If this flag is not set, the regcomp subroutine sets the re_nsub structure to the number of parenthetic expressions found in the Pattern parameter.
  - **REG_NEWLINE**: Prohibits . (period) and nonmatching bracket expression from matching a new-line character. The ^ (circumflex) and $ (dollar sign) will match the zero-length string immediately following or preceding a new-line character.

Return Values
If successful, the regcomp subroutine returns a value of 0. Otherwise, it returns another value indicating the type of failure, and the content of the Preg parameter is undefined.
Error Codes

The following macro names for error codes may be written to the errno global variable under error conditions:

- **REG_BADPAT**: Indicates a basic or extended regular expression that is not valid.
- **REG_ECOLLATE**: Indicates a collating element referenced that is not valid.
- **REG_ECTYPE**: Indicates a character class-type reference that is not valid.
- **REG_EESCAPE**: Indicates a trailing \ in pattern.
- **REG_ESUBREG**: Indicates a number in digit is not valid or in error.
- **REG_EBRACK**: Indicates a [] imbalance.
- **REG_EPAREN**: Indicates a ( ) or ( ) imbalance.
- **REG_EBRACE**: Indicates a { } imbalance.
- **REG_BADBR**: Indicates the content of { } is unusable: not a number, number too large, more than two numbers, or first number larger than second.
- **REG_ERANGE**: Indicates an unusable end point in range expression.
- **REG_ESPACE**: Indicates out of memory.
- **REG_BADRPT**: Indicates a ? (question mark), * (asterisk), or + (plus sign) not preceded by valid basic or extended regular expression.

If the regcomp subroutine detects an illegal basic or extended regular expression, it can return either the REG_BADPAT error code or another that more precisely describes the error.

Examples

The following example illustrates how to match a string (specified in the string parameter) against an extended regular expression (specified in the Pattern parameter):

```c
#include <sys/types.h>
#include <regex.h>

int match(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);
}
```

In the preceding example, errors are treated as no match. When there is no match or error, the calling process can get details by calling the regerror subroutine.

Related Information

The regerror (regerror Subroutine on page 46) subroutine, regexec (regexec Subroutine on page 47) subroutine, regfree (regfree Subroutine on page 50) subroutine.

Subroutines Overview and Understanding Internationalized Regular Expression Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
regerror Subroutine

Purpose
Returns a string that describes the _ErrCode_ parameter.

Library
Standard C Library (_libc.a_)

Syntax
```c
#include <regex.h>

size_t regerror (ErrCode, Preg, ErrBuf, ErrBuf_Size)
int  * ErrCode;
const regex_t * Preg;
char *  ErrBuf;
size_t  ErrBuf_Size;
```

Description
The `regerror` subroutine provides a mapping from error codes returned by the `regcomp` and `regexec` subroutines to printable strings. It generates a string corresponding to the value of the _ErrCode_ parameter, which is the last nonzero value returned by the `regcomp` or `regexec` subroutine with the given value of the _Preg_ parameter. If the _ErrCode_ parameter is not such a value, the content of the generated string is unspecified. The string generated is obtained from the `regex.cat` message catalog.

If the _ErrBuf_Size_ parameter is not 0, the `regerror` subroutine places the generated string into the buffer specifier by the _ErrBuf_ parameter, whose size in bytes is specified by the _ErrBuf_Size_ parameter. If the string (including the terminating null character) cannot fit in the buffer, the `regerror` subroutine truncates the string and null terminates the result.

Parameters
- **ErrCode**: Specifies the error for which a description string is to be returned.
- **Preg**: Specifies the structure that holds the previously compiled output of the `regcomp` subroutine.
- **ErrBuf**: Specifies the buffer to receive the string generated by the `regerror` subroutine.
- **ErrBuf_Size**: Specifies the size of the _ErrBuf_ parameter.

Return Values
The `regerror` subroutine returns the size of the buffer needed to hold the entire generated string, including the null termination. If the return value is greater than the value of the _ErrBuf_Size_ variable, the string returned in the _ErrBuf_ buffer is truncated.

Error Codes
If the _ErrBuf_Size_ value is 0, the regerror subroutine ignores the _ErrBuf_ parameter, but returns the one of the following error codes. These error codes defined in the regex.h file.

- **REG_NOMATCH**: Indicates the basic or extended regular expression was unable to find a match.
- **REG_BADPAT**: Indicates a basic or extended regular expression that is not valid.
- **REG_ECOLLATE**: Indicates a collating element referenced that is not valid.
- **REG_ECTYPE**: Indicates a character class-type reference that is not valid.
- **REG_EESCAPE**: Indicates a trailing \ in pattern.
- **REG_ESUBREG**: Indicates a number in \digit is not valid or in error.
REG_EBRACK Indicates a [] imbalance.
REG_EPAREN Indicates a () or () imbalance.
REG_EBRACE Indicates a {} imbalance.
REG_BADBR Indicates the content of {} is unusable: not a number, number too large, more than two numbers, or first number larger than second.
REG_ERANGE Indicates an unusable end point in range expression.
REG_ESPACE Indicates out of memory.
REG_BADRPT Indicates a ? (question mark), * (asterisk), or + (plus sign) not preceded by valid basic or extended regular expression.
REG_ENEWLINE Indicates a new-line character was found before the end of the regular or extended regular expression, and REG_NEWLINE was not set.

If the Preg parameter passed to the regexec subroutine is not a compiled basic or extended regular expression returned by the regcomp subroutine, the result is undefined.

Examples
An application can use the regerror subroutine (with the parameters (Code, Preg, null, (size_t) 0) passed to it) to determine the size of buffer needed for the generated string, call the malloc subroutine to allocate a buffer to hold the string, and then call the regerror subroutine again to get the string. Alternately, this subroutine can allocate a fixed, static buffer that is large enough to hold most strings (perhaps 128 bytes), and then call the malloc subroutine to allocate a larger buffer if necessary.

Related Information
The regcomp (“regcomp Subroutine” on page 44) subroutine, regexec (“regexec Subroutine”) subroutine, regfree (“regfree Subroutine” on page 50) subroutine.

Subroutines Overview and Understanding Internationalized Regular Expression Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

regexec Subroutine

Purpose
Compares the null-terminated string specified by the value of the String parameter against the compiled basic or extended regular expression Preg, which must have previously been compiled by a call to the regcomp subroutine.

Library
Standard C Library (libc. a)

Syntax
#include <regex.h>

int regexec (Preg, String, NMatch, PMatch, EFlags)
const regex_t * Preg;
const char * String;
size_t NMatch;
regmatch_t * PMatch;
int EFlags;
Description

The `regexec` subroutine compares the null-terminated string in the `String` parameter with the compiled basic or extended regular expression in the `Preg` parameter initialized by a previous call to the `regcomp` subroutine. If a match is found, the `regexec` subroutine returns a value of 0. The `regexec` subroutine returns a nonzero value if it finds no match or it finds an error.

If the `NMatch` parameter has a value of 0, or if the `REG_NOSUB` flag was set on the call to the `regcomp` subroutine, the `regexec` subroutine ignores the `PMatch` parameter. Otherwise, the `PMatch` parameter points to an array of at least the number of elements specified by the `NMatch` parameter. The `regexec` subroutine fills in the elements of the array pointed to by the `PMatch` parameter with offsets of the substrings of the `String` parameter. The offsets correspond to the parenthetic subexpressions of the original `pattern` parameter that was specified to the `regcomp` subroutine.

The `pmatch.rm_so` structure is the byte offset of the beginning of the substring, and the `pmatch.rm_eo` structure is one greater than the byte offset of the end of the substring. Subexpression $i$ begins at the $i$th matched open parenthesis, counting from 1. The 0 element of the array corresponds to the entire pattern. Unused elements of the `PMatch` parameter, up to the value `PMatch[NMatch-1]`, are filled with -1. If more than the number of subexpressions specified by the `NMatch` parameter (the `pattern` parameter itself counts as a subexpression), only the first `NMatch-1` subexpressions are recorded.

When a basic or extended regular expression is being matched, any given parenthetic subexpression of the `pattern` parameter might match several different substrings of the `String` parameter. Otherwise, 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 the `PMatch` parameter when regular expressions are matched:

- If a subexpression in a regular expression participated in the match several times, the offset of the last matching substring is reported in the `PMatch` parameter.
- If a subexpression did not participate in a match, the byte offset in the `PMatch` parameter is a value of -1. A subexpression does not participate in a match if any of the following are true:
  - An * (asterisk) or \{\} (backslash, left brace, backslash, right brace) appears immediately after the subexpression in a basic regular expression.
  - An * (asterisk), ? (question mark), or \{} (left and right braces) appears immediately after the subexpression in an extended regular expression and the subexpression did not match (matched 0 times).
  - A | (pipe) is used in an extended regular expression to select either the subexpression that didn’t match or another subexpression, and the other subexpression matched.
- If a subexpression is contained in a subexpression, the data in the `PMatch` parameter refers to the last such subexpression.
- If a subexpression is contained in a subexpression and the byte offsets in the `PMatch` parameter have a value of -1, the pointers in the `PMatch` parameter also have a value of -1.
- If a subexpression matched a zero-length string, the offsets in the `PMatch` parameter refer to the byte immediately following the matching string.

If the `REG_NOSUB` flag was set in the `cflags` parameter in the call to the `regcomp` subroutine, and the `NMatch` parameter is not equal to 0 in the call to the `regexec` subroutine, the content of the `PMatch` array is unspecified.

If the `REG_NEWLINE` flag was not set in the `cflags` parameter when the `regcomp` subroutine was called, then a new-line character in the `pattern` or `String` parameter is treated as an ordinary character. If the `REG_NEWLINE` flag was set when the `regcomp` subroutine was called, the new-line character is treated as an ordinary character except as follows:
A new-line character in the String parameter is not matched by a period outside of a bracket expression or by any form of a nonmatching list. A nonmatching list expression begins with a ^ (circumflex) and specifies a list that matches any character or collating element and the expression in the list after the leading caret. For example, the regular expression [^
abc] matches any character except a, b, or c. The circumflex has this special meaning only when it is the first character in the list, immediately following the left bracket.

A ^ (circumflex) in the pattern parameter, when used to specify expression anchoring, matches the zero-length string immediately after a new-line character in the String parameter, regardless of the setting of the REG_NOTBOL flag.

A $ (dollar sign) in the pattern parameter, when used to specify expression anchoring, matches the zero-length string immediately before a new-line character in the String parameter, regardless of the setting of the REG_NOTEOL flag.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preg</td>
<td>Contains the compiled basic or extended regular expression to compare against the String parameter.</td>
</tr>
<tr>
<td>String</td>
<td>Contains the data to be matched.</td>
</tr>
<tr>
<td>NMatch</td>
<td>Contains the number of subexpressions to match.</td>
</tr>
<tr>
<td>PMatch</td>
<td>Contains the array of offsets into the String parameter that match the corresponding subexpression in the Preg parameter.</td>
</tr>
<tr>
<td>EFlags</td>
<td>Contains the bitwise inclusive OR of 0 or more of the flags controlling the behavior of the regexexec subroutine capable of customizing.</td>
</tr>
</tbody>
</table>

The EFlags parameter modifies the interpretation of the contents of the String parameter. It is the bitwise inclusive OR of 0 or more of the following flags, which are defined in the regex.h file:

- **REG_NOTBOL**
  - The first character of the string pointed to by the String parameter is not the beginning of the line. Therefore, the ^ (circumflex), when used as a special character, does not match the beginning of the String parameter.

- **REG_NOTEOL**
  - The last character of the string pointed to by the String parameter is not the end of the line. Therefore, the $ (dollar sign), when used as a special character, does not match the end of the String parameter.

**Return Values**

On successful completion, the regexexec subroutine returns a value of 0 to indicate that the contents of the String parameter matched the contents of the pattern parameter, or to indicate that no match occurred. The REG_NOMATCH error is defined in the regex.h file.

**Error Codes**

If the regexexec subroutine is unsuccessful, it returns a nonzero value indicating the type of problem. The following macros for possible error codes that can be returned are defined in the regex.h file:

- **REG_NOMATCH** Indicates the basic or extended regular expression was unable to find a match.
- **REG_BADPAT** Indicates a basic or extended regular expression that is not valid.
- **REG_ECOLLATE** Indicates a collating element referenced that is not valid.
- **REG_ECTYPE** Indicates a character class-type reference that is not valid.
- **REG_EESCAPE** Indicates a trailing \ (backslash) in the pattern.
- **REG_ESUBREG** Indicates a number in \digit is not valid or is in error.
- **REG_EBRACK** Indicates a \[ \] (left and right brackets) imbalance.
- **REG_EPAREN** Indicates a \( \) (backslash, left parenthesis, backslash, right parenthesis) or ( ) (left and right parentheses) imbalance.
- **REG_EBRACE** Indicates a \{ \} (backslash, left brace, backslash, right brace) imbalance.
REG_BADBR Indicates the content of `\{\}` (backslash, left brace, backslash, right brace) is unusable (not a number, number too large, more than two numbers, or first number larger than second).

REG_ERANGE Indicates an unusable end point in range expression.

REG_ESPACE Indicates out of memory.

REG_BADRPT Indicates a `?` (question mark), `*` (asterisk), or `+` (plus sign) not preceded by valid basic or extended regular expression.

If the value of the `Preg` parameter to the `regexec` subroutine is not a compiled basic or extended regular expression returned by the `regcomp` subroutine, the result is undefined.

Examples

The following example demonstrates how the `REG_NOTBOL` flag can be used with the `regexec` subroutine to find all substrings in a line that match a pattern supplied by a user. (For simplicity, very little error-checking is done in this example.)

```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.r_sp and pm.rm_ep>
  /* This call to regexec finds the next match */
  error = regexec (&re, pm.rm_ep, 1, &pm, REG_NOTBOL);
```

Related Information

The `regcomp` subroutine, `regerror` subroutine, `regfree` subroutine.

Subroutines Overview and Understanding Internationalized Regular Expression Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

regfree Subroutine

Purpose

Frees any memory allocated by the `regcomp` subroutine associated with the `Preg` parameter.

Library

Standard C Library (libc.a)

Syntax

```c
#include <regex.h>

void regfree (Preg)
regex_t *Preg;
```

Description

The `regfree` subroutine frees any memory allocated by the `regcomp` subroutine associated with the `Preg` parameter. An expression defined by the `Preg` parameter is no longer treated as a compiled basic or extended regular expression after it is given to the `regfree` subroutine.

Parameters

`Preg` Structure containing the compiled output of the `regcomp` subroutine. Memory associated with this structure is freed by the `regfree` subroutine.
Related Information
The `regcomp` subroutine, `regerror` subroutine, `regexec` subroutine.

Subroutines Overview and Understanding Internationalized Regular Expression Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

`reltimerid` Subroutine

Purpose
Releases a previously allocated interval timer.

Library
Standard C Library (`libc.a`)

Syntax
```c
#include <sys/time.h>
#include <sys/events.h>

int reltimerid (TimerID);

timer_t TimerID;
```

Description
The `reltimerid` subroutine is used to release a previously allocated interval timer, which is returned by the `gettimerid` subroutine. Any pending timer event generated by this interval timer is cancelled when the call returns.

Parameters

* `TimerID` Specifies the ID of the interval timer being released.

Return Values
The `reltimerid` subroutine returns a 0 if it is successful. If an error occurs, the value -1 is returned and `errno` is set.

Error Codes
If the `reltimerid` subroutine fails, a -1 is returned and `errno` is set with the following error code:

* `EINVAL` The timer ID specified by the `Timerid` parameter is not a valid timer ID.

Related Information
The `gettimerid` subroutine.


Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
remainder, remainderf, or remainderl Subroutine

Purpose
Returns the floating-point remainder.

Syntax
#include <math.h>

double remainder (x, y)
double x;
double y;

float remainderf (x, y)
float x;
float y;

long double remainderl (x, y)
long double x;
long double y;

Description
The remainder, remainderf, and remainderl subroutines return the floating-point remainder \( r = x - ny \) when \( y \) is nonzero. The value \( n \) is the integral value nearest the exact value \( x/y \). When \( |n| \times y = \frac{1}{2} \), the value \( n \) is chosen to be even.

Parameters
\( x \) specifies the value of the numerator.
\( y \) specifies the value of the denominator.

Return Values
Upon successful completion, the remainder, remainderf, and remainderl subroutines return the floating-point remainder \( r = x - ny \) when \( y \) is nonzero.

If \( x \) or \( y \) is NaN, a NaN is returned.

If \( x \) is infinite or \( y \) is 0 and the other is non-NaN, a domain error occurs, and a NaN is returned.

Related Information

math.h in AIX 5L Version 5.3 Files Reference.

remove Subroutine

Purpose
Removes a file.

Library
Standard C Library (libc.a)
Syntax

```c
#include <stdio.h>

int remove(const char *FileName);
```

Description

The `remove` subroutine makes a file named by `FileName` inaccessible by that name. An attempt to open that file using that name does not work unless you recreate it. If the file is open, the subroutine does not remove it.

If the file designated by the `FileName` parameter has multiple links, the link count of files linked to the removed file is reduced by 1.

Parameters

- `FileName` Specifies the name of the file being removed.

Return Values

Upon successful completion, the `remove` subroutine returns a value of 0; otherwise it returns a nonzero value.

Related Information

- The `link` subroutine, `rename` subroutine.
- The `unlink` command.

removeea Subroutine

Purpose

Removes an extended attribute.

Syntax

```c
#include <sys/ea.h>

int removeea(const char *path, const char *name);
int fremoveea(int filedes, const char *name);
int lremoveea(const char *path, const char *name);
```

Description

Extended attributes are name:value pairs associated with the file system objects (such as files, directories, and symlinks). They are extensions to the normal attributes that are associated with all objects in the file system (that is, the `stat(2)` data).

The `removeea` subroutine removes the extended attribute identified by `name` and associated with the given `path` in the file system. The `fremoveea` subroutine is identical to `removeea`, except that it takes a file descriptor instead of a path. The `lremoveea` subroutine is identical to `removeea`, except, in the case of a symbolic link, the link itself is interrogated rather than the file that it refers to.
Parameters

- **path**: The path name of the file.
- **name**: The name of the extended attribute. An extended attribute name is a NULL-terminated string.
- **filedes**: A file descriptor for the file.

Return Values

If the `removeea` subroutine succeeds, 0 is returned. Upon failure, -1 is returned and **errno** is set appropriately.

Error Codes

- **EACCES**: Caller lacks write permission on the base file, or lacks the appropriate ACL privileges for named attribute **delete**.
- **EFAULT**: A bad address was passed for **path** or **name**.
- **EFORMAT**: File system is capable of supporting EAs, but EAs are disabled.
- **EINVAL**: A path-like name should not be used (such as **zml/file**, ., and ..).
- **ENOATTR**: The named attribute does not exist, or the process has no access to this attribute.
- **ENOTSUP**: Extended attributes are not supported by the file system.

Related Information

- `getea Subroutine`
- `listea Subroutine`
- "`setea Subroutine`" on page 166
- "`statea Subroutine`" on page 314

remquo, remquof, or remquol Subroutine

Purpose

Returns the floating-point remainder.

Syntax

```c
#include <math.h>

double remquo (x, y, quo);
double x;
double y;
int *quo;

float remquof (x, y, quo);
float x;
float y;
int *quo;

long double remquol (x, y, quo);
long double x;
long double y;
int *quo;
```

Description

The `remquo`, `remquof`, and `remquol` subroutines compute the same remainder as the `remainder`, `remainderf`, and `remainderl` functions, respectively. In the object pointed to by **quo**, they store a value whose sign is the sign of **x/y** and whose magnitude is congruent modulo $2^n$ to the magnitude of the integral quotient of **x/y**, where **n** is 3.
An application wishing to check for error situations should set the `errno` global variable to zero and call `feclearexcept(FE_ALL_EXCEPT)` before calling these subroutines. Upon return, if `errno` is nonzero or `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is nonzero, an error has occurred.

**Parameters**

- **x** Specifies the value of the numerator.
- **y** Specifies the value of the denominator.
- **quo** Points to the object where a value whose sign is the sign of \(x/y\) is stored.

**Return Values**

The `remquo`, `remquof`, and `remquol` subroutines return \(x \text{ REM } y\).

If \(x\) or \(y\) is NaN, a NaN is returned.

If \(x\) is ±Inf or \(y\) is zero and the other argument is non-NaN, a domain error occurs, and a NaN is returned.

**Related Information**

- "remainder, remainderf, or remainderl Subroutine" on page 52
- `math.h` in AIX 5L Version 5.3 Files Reference.

---

**rename Subroutine**

**Purpose**

Renames a directory or a file.

**Library**

Standard C Library (`libc.a`)

**Syntax**

```c
#include <stdio.h>

int rename (const char *FromPath, const char *ToPath);
```

**Description**

The `rename` subroutine renames a directory or a file within a file system.

To use the `rename` subroutine, the calling process must have write and search permission in the parent directories of both the `FromPath` and `ToPath` parameters. If the path defined in the `FromPath` parameter is a directory, the calling process must have write and search permission to the `FromPath` directory as well.

If the `FromPath` and `ToPath` parameters both refer to the same existing file, the `rename` subroutine returns successfully and performs no other action.
The components of both the FromPath and ToPath parameters must be of the same type (that is, both directories or both non-directories) and must reside on the same file system. If the ToPath file already exists, it is first removed. Removing it guarantees that a link named ToPath will exist throughout the operation. This link refers to the file named by either the ToPath or FromPath parameter before the operation began.

If the final component of the FromPath parameter is a symbolic link, the symbolic link (not the file or directory to which it points) is renamed. If the ToPath is a symbolic link, the link is destroyed.

If the parent directory of the FromPath parameter has the Sticky bit attribute (described in the sys/mode.h file), the calling process must have an effective user ID equal to the owner ID of the FromPath parameter, or to the owner ID of the parent directory of the FromPath parameter.

A user who is not the owner of the file or directory must have root user authority to use the rename subroutine.

If the FromPath and ToPath parameters name directories, the following must be true:
- The directory specified by the FromPath parameter is not an ancestor of ToPath. For example, the FromPath path name must not contain a path prefix that names the directory specified by the ToPath parameter.
- The directory specified in the FromPath parameter must be well-formed. A well-formed directory contains both . (dot) and .. (dot dot) entries. That is, the . (dot) entry in the FromPath directory refers to the same directory as that in the FromPath parameter. The .. (dot dot) entry in the FromPath directory refers to the directory that contains an entry for FromPath.
- The directory specified by the ToPath parameter, if it exists, must be well-formed (as defined previously).

Parameters

FromPath
Identifies the file or directory to be renamed.

ToPath
Identifies the new path name of the file or directory to be renamed. If ToPath is an existing file or empty directory, it is replaced by FromPath. If ToPath specifies a directory that is not empty, the rename subroutine exits with an error.

Return Values

Upon successful completion, the rename subroutine returns a value of 0. Otherwise, a value of -1 is returned, and the errno global variable is set to indicate the error.

Error Codes

The rename subroutine is unsuccessful and the file or directory name remains unchanged if one or more of the following are true:

EACCES
Creating the requested link requires writing in a directory mode that denies the process write permission.

EBUSY
The directory named by the FromPath or ToPath parameter is currently in use by the system, or the file named by FromPath or ToPath is a named STREAM.

EDQUOT
The directory that would contain the path specified by the ToPath parameter cannot be extended because the user’s or group’s quota of disk blocks on the file system containing the directory is exhausted.

EEXIST
The ToPath parameter specifies an existing directory that is not empty.

EINVAL
The path specified in the FromPath or ToPath parameter is not a well-formed directory (FromPath is an ancestor of ToPath), or an attempt has been made to rename . (dot) or .. (dot dot).
The `rename` subroutine can be unsuccessful if the following is true:

- **ETIMEDOUT** The connection timed out.

The `rename` subroutine can be unsuccessful for other reasons. See Appendix A, “Base Operating System Error Codes For Services That Require Path-Name Resolution” for a list of additional errors.

### Related Information

The `chmod` subroutine, `link` subroutine, `mkdir` subroutine, `rmdir` subroutine, `unlink` subroutine.

The `chmod` command, `mkdir` command, `mv` command, `mvdir` command.

[Files, Directories, and File Systems for Programmers](#) in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

### reset_malloc_log Subroutine

#### Purpose

Resets information collected by the malloc subsystem.

#### Syntax

```c
#include <malloc.h>
void reset_malloc_log (addr)
void *addr;
```

#### Description

The `reset_malloc_log` subroutine resets the record of currently active malloc allocations stored by the malloc subsystem. These records are stored in `malloc_log` structures, which are located in the process heap. Only records corresponding to the heap of which `addr` is a member are reset, unless `addr` is NULL, in which case records for all heaps are reset. The `addr` parameter must be a pointer to space allocated previously by the malloc subsystem or NULL, otherwise no information is reset and the `errno` global variable is set to `EINVAL`. 
Parameters

addr  Pointer to space allocated previously by the malloc subsystem

Related Information
malloc Subroutine, get_malloc_log Subroutine, and get_malloc_log_live Subroutine in AIX 5L Version 5.3
Technical Reference: Base Operating System and Extensions Volume 1

revoke Subroutine

Purpose
Revoke access to a file.

Library
Standard C Library (libc.a)

Syntax
int revoke (char *Path);

Description
The revoke subroutine revokes access to a file by all processes.

All accesses to the file are revoked. Subsequent attempts to access the file using a file descriptor established before the revoke subroutine fail and cause the process to receive a return value of -1, and the errno global variable is set to EBADF.

A process can revoke access to a file only if its effective user ID is the same as the file owner ID, or if the calling process is privileged.

Note: The revoke subroutine has no affect on subsequent attempts to open the file. To assure exclusive access to the file, the caller should change the access mode of the file before issuing the revoke subroutine. Currently the revoke subroutine works only on terminal devices. The chmod subroutine changes file access modes.

Parameters
Path  Path name of the file for which access is to be revoked.

Return Values
Upon successful completion, the revoke subroutine returns a value of 0.

If the revoke subroutine fails, a value of -1 returns and the errno global variable is set to indicate the error.

Error Codes
The revoke subroutine fails if any of the following are true:

ENOTDIR  A component of the path prefix is not a directory.
EACCES  Search permission is denied on a component of the path prefix.
ENOENT
A component of the path prefix does not exist, or the process has the disallow truncation attribute (see the ulimit subroutine).
ENOENT
The path name is null.
ENOENT
A symbolic link was named, but the file to which it refers does not exist.
ESTALE
The process’s root or current directory is located in a virtual file system that has been unmounted.
EFAULT
The Path parameter points outside of the process’s address space.
ELOOP
Too many symbolic links were encountered in translating the path name.
ENAMETOOLONG
A component of a path name exceeds 255 characters, or an entire path name exceeds 1023 characters.
EIO
An I/O error occurred during the operation.
EPERM
The effective user ID of the calling process is not the same as the file’s owner ID.
EINVAL
Access rights revocation is not implemented for this file.

Related Information
The chmod subroutine, frevoke subroutine.

List of Security and Auditing Subroutines and Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

rintf, rintl, or rint Subroutine

Purpose
Rounds to the nearest integral value.

Syntax
#include <math.h>

float rintf (x)
float x;

long double rintl (x)
long double x;

double rint (x)
double x;

Description
The rintf, rintl, and rint subroutines return the integral value (represented as a double) nearest x in the direction of the current rounding mode. The current rounding mode is implementation-defined.

The rintf, rintl, and rint subroutines differ from the nearbyint, nearbyintf, and nearbyintl subroutines only in that they may raise the inexact floating-point exception if the result differs in value from the argument.

An application wishing to check for error situations should set the errno global variable to zero and call feclearexcept(FE_ALL_EXCEPT) before calling these subroutines. Upon return, if errno is nonzero or fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is nonzero, an error has occurred.

Parameters

x 
Specifies the value to be rounded.
**Return Values**

Upon successful completion, the `rintf`, `rintl`, and `rint` subroutines return the integer (represented as a double precision number) nearest \( x \) in the direction of the current rounding mode.

If \( x \) is NaN, a NaN is returned.

If \( x \) is ±0 or ±Inf, \( x \) is returned.

If the correct value would cause overflow, a range error occurs the `rintf`, `rintl`, and `rint` subroutines return the value of the macro ±`HUGE_VALF` and ±`HUGE_VALL` (with the same sign as \( x \)), respectively.

**Related Information**

`abs` Subroutine, `floor`, `floorl`, `ceil`, `ceill`, `nearest`, `trunc`, `rint`, `itrunc`, `uitrunc`, `fmod`, `fmodl`, `fabs`, or `fabsl` Subroutines, and `lldiv` Subroutine in AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 1.

`math.h` in AIX 5L Version 5.3 Files Reference.

---

**rmdir Subroutine**

**Purpose**

Removes a directory.

**Library**

Standard C Library (`libc.a`)

**Syntax**

```c
#include <unistd.h>

int rmdir (const char *Path);
```

**Description**

The `rmdir` subroutine removes the directory specified by the `Path` parameter. If Network File System (NFS) is installed on your system, this path can cross into another node.

For the `rmdir` subroutine to execute successfully, the calling process must have write access to the parent directory of the `Path` parameter.

In addition, if the parent directory of `Path` has the Sticky bit attribute (described in the `sys/mode.h` file), the calling process must have one of the following:

- An effective user ID equal to the directory to be removed
- An effective user ID equal to the owner ID of the parent directory of `Path`
- Root user authority.
Parameters

Path  Specifies the directory path name. The directory you specify must be:

   Empty   The directory contains no entries other than . (dot) and .. (dot dot).

   Well-formed
   If the . (dot) entry in the Path parameter exists, it must refer to the same directory as Path. Exactly
   one directory has a link to the Path parameter, excluding the self-referential . (dot). If the .. (dot
dot) entry in Path exists, it must refer to the directory that contains an entry for Path.

Return Values

Upon successful completion, the rmdir subroutine returns a value of 0. Otherwise, a value of -1 is
returned, the specified directory is not changed, and the errno global variable is set to indicate the error.

Error Codes

The rmdir subroutine fails and the directory is not deleted if the following errors occur:

EACCESS  There is no search permission on a component of the path prefix, or there is no
   write permission on the parent directory of the directory to be removed.

EBUSY    The directory is in use as a mount point.

EEXIST or ENOTEMPTY  The directory named by the Path parameter is not empty.

ENAMETOOLONG  The length of the Path parameter exceeds PATH_MAX; or a path-name
   component longer than NAME_MAX and POSIX_NO_TRUNC is in effect.

ENOENT   The directory named by the Path parameter does not exist, or the Path
   parameter points to an empty string.

ENOTDIR  A component specified by the Path parameter is not a directory.

EINVAL  The directory named by the Path parameter is not well-formed.

EROFS   The directory named by the Path parameter resides on a read-only file system.

The rmdir subroutine can be unsuccessful for other reasons. See Appendix A, “Base Operating System
Error Codes For Services That Require Path-Name Resolution” on page A-1 for a list of additional errors.

If NFS is installed on the system, the rmdir subroutine fails if the following is true:

ETIMEDOUT  The connection timed out.

Related Information

The chmod or fchmod subroutine, mkdir subroutine, remove ("remove Subroutine" on page 52)
subroutine, rename ("rename Subroutine" on page 55) subroutine, umask ("umask Subroutine" on page
466) subroutine, unlink ("unlink Subroutine" on page 471) subroutine.

The rm command. rmdir command.

Files, Directories, and File Systems For Programmers in AIX 5L Version 5.3 General Programming
Concepts: Writing and Debugging Programs.

rmproj Subroutine

Purpose

Removes project definition from kernel project registry.
Library
The libacct.a library.

Syntax
<sys/aacct.h>

rmproj(struct project *, int flag)

Description
The rmproj subroutine removes the definition of a project from kernel project registry. It takes a pointer to project structure as input argument that holds the name or number of a project that needs to be removed. The flag is set to indicate whether a name or number is supplied as input, as follows:

- PROJ_NAME — Indicates that the supplied project definition only has the project name. The rmproj subroutine queries the kernel to obtain a match for the supplied project name and returns the matching entry.
- PROJ_NUM — Indicates that the supplied project definition only has the project number. The rmproj subroutine queries the kernel to obtain a match for the supplied project number and returns the matching entry.

Parameters

project Pointer holding the details of the project to be removed.
flag An integer flag which indicates whether the supplied project definition structure has project name and number that need to be removed.

Security
Only for privileged users. Privilege can be extended to nonroot users by granting the CAP_AACCT capability to a user.

Return Values

0 Success
-1 Failure

Error Codes
EINVAL Pointer is null or the flag parameter is set to an invalid value.
ENOENT Project Definition does not exist.
EPERM Permission denied.

Related Information
The addproj Subroutine, chprojattr Subroutine, getproj Subroutine, getprojs Subroutine, "rmprojdb Subroutine."

rmprojdb Subroutine

Purpose
Removes the specified project definition from the specified project database.
Library
The libaacct.a library.

Syntax
<syntax lang="c">
rmprojdb(void *handle, struct project *project, int flag)
</syntax>

Description
The rmprojdb subroutine removes the project definition stored in the struct project variable from the project named by the handle parameter. The project database must be initialized before calling this subroutine. The projdballoc and projdbfinit subroutines are provided for this purpose. If the supplied project definition does not exist in the named project database, the rmprojdb subroutine returns -1 and sets errno to ENOENT.

The rmprojdb subroutine takes a pointer to a project structure as an input argument. This pointer to the project structure holds the name or number of a project that needs to be removed. The flag parameter is set to indicate whether a name or number is supplied as input as follows:

- PROJ_NAME — Indicates that the supplied project definition only has the project name.
- PROJ_NUM — Indicates that the supplied project definition only has the project number.

There is an internal state (that is, the current project) associated with the project database. When the project database is initialized, the current project is the first project in the database. The rmprojdb subroutine removes the named project and repositions the internal current project to the first project definition.

Parameters

- handle: Pointer to project database handle.
- project: Pointer to a project structure that holds the definition of the project to be added.
- flag: Integer flag to indicate whether the name or number of the project is supplied.

Security
Only for privileged users. Privilege can be extended to nonroot users by granting the CAP_AACCT capability to a user.

Return Values

- 0: Success
- -1: Failure

Error Codes

- ENOENT: Project definition does not exist
- EPERM: Permission denied. The user is not a privileged user.
- EINVAL: Passed pointer is NULL or the flag parameter holds an invalid value.

Related Information

The addprojdb Subroutine, chprojattrdb Subroutine, getfirstprojdb Subroutine, getnextprojdb Subroutine, getprojdb Subroutine, projdballoc Subroutine, projdbfinit Subroutine, projdbfree Subroutine, "rmprojdb Subroutine" on page 61.
round, roundf, or roundl Subroutine

Purpose
Rounds to the nearest integer value in a floating-point format.

Syntax
```c
#include <math.h>

double round (double x);
float roundf (float x);
long double roundl (long double x);
```

Description
The `round`, `roundf`, and `roundl` subroutines round the `x` parameter to the nearest integer value in floating-point format, rounding halfway cases away from zero, regardless of the current rounding direction.

An application wishing to check for error situations should set the `errno` global variable to zero and call `feclearexcept(FE_ALL_EXCEPT)` before calling these subroutines. Upon return, if `errno` is nonzero or `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is nonzero, an error has occurred.

Parameters
- `x` Specifies the value to be rounded.

Return Values
Upon successful completion, the `round`, `roundf`, and `roundl` subroutines return the rounded integer value.

If `x` is NaN, a NaN is returned.

If `x` is ±0 or ±Inf, `x` is returned.

If the correct value would cause overflow, a range error occurs and the `round`, `roundf`, and `roundl` subroutines return the value of the macro ±`HUGE_VAL`, ±`HUGE_VALF`, and ±`HUGE_VALL` (with the same sign as `x`), respectively.

Related Information
- `math.h` in AIX 5L Version 5.3 Files Reference.

rpmatch Subroutine

Purpose
Determines whether the response to a question is affirmative or negative.
Library
Standard C Library (libc.a)

Syntax
#include <stdlib.h>

int rpmatch (const char *Response);

Description
The `rpmatch` subroutine determines whether the expression in the `Response` parameter matches the affirmative or negative response specified by the `LC_MESSAGES` category in the current locale. Both expressions can be extended regular expressions.

Parameters
- `Response`: Specifies input entered in response to a question that requires an affirmative or negative reply.

Return Values
This subroutine returns a value of 1 if the expression in the `Response` parameter matches the locale’s affirmative expression. It returns a value of 0 if the expression in the `Response` parameter matches the locale’s negative expression. If neither expression matches the expression in the `Response` parameter, a -1 is returned.

Examples
The following example shows an affirmative expression in the En_US locale. This example matches any expression in the `Response` parameter that begins with a `y` or `Y` followed by zero or more alphabetic characters, or it matches the letter `o` followed by the letter `k`.

```
^[yY][[:alpha:]]* | ok
```

Related Information
The `localeconv`, `nl_langinfo`, `regcomp` subroutine, `regexec` subroutine, `setlocale` subroutine.


Subroutines, Example Programs, and Libraries in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

RSiAddSetHot Subroutine

Purpose
Add a single set of peer statistics to an already defined SpmiHotSet

Library
RSI Library (libSpmi.a)
```
#include sys/Rsi.h

struct SpmiHotVals *RSiAddSetHot(
    rhandle, HotSet, StatName,
    GrandParent,
    maxresp, threshold, frequency, feed_type,
    except_type, severity, trap_no)

RSiHandle rhandle;
struct SpmiHotSet *HotSet;
char *StatName;
cx_handle GrandParent;
int maxresp;
int threshold;
int frequency;
int feed_type;
int except_type;
int severity;
int trap_no;
```

### Parameters

- **rhandle**: Must be an `RSiHandle`, which was previously initialized by the `RSiOpen` subroutine.
- **HotSet**: Specifies a pointer to a valid structure of type `SpmiHotSet` as created by the `RSiCreateHotSet` subroutine call.
- **StatName**: Specifies the name of the statistic within the subcontexts (peer contexts) of the context identified by the `GrandParent` parameter.
- **GrandParent**: Specifies a valid `cx_handle` handle as obtained by another subroutine call. The handle must identify a context with at least one subcontext, which contains the statistic identified by the `StatName` parameter. If the context specified is one of the `RTime` contexts, no subcontext need to be created at the time the `SpmiAddSetHot` subroutine call is issued; the presence of the metric identified by the `StatName` parameter is checked against the context class description.

If the context specified has or may have multiple levels of instantiable contexts below it (such as the `FS` and `RTime/ARM` contexts), the metric is only searched for at the lowest context level. The `SpmiHotSet` created is a pseudo hotvals structure used to link together a peer group of `SpmiHotVals` structures, which are created under the covers, one for each subcontext of the `GrandParent` context. In the case of `RTime/ARM`, if additional contexts are later added under the `GrandParent` contexts, additional hotsets are added to the peer group. This is transparent to the application program, except that the `RSiGetHotItem` subroutine call will return the peer group `SpmiHotVals` pointer rather than the pointer to the pseudo structure.

Note that specifying a specific volume group context (such as `FS/rootvg`) or a specific application context (such as `RTime/ARN/arnpeek`) is still valid and won’t involve creation of pseudo `SpmiHotVals` structures.

- **maxresp**: Must be non-zero if `except_type` specifies that exceptions or SNMP traps must be generated. If specified as zero, indicates that all `SpmiHotItems` that meet the criteria specified by `threshold` must be returned, up-to a maximum of `maxresp` items. If both exceptions/traps and feeds are requested, the `maxresp` value is used to cap the number of exceptions/alerts as well as the number of items returned. If `feed_type` is specified as `SiHotAlways`, the `maxresp` parameter is still used to return at most `maxresp` items.

Where the `GrandParent` argument specifies a context that has multiple levels of instantiable contexts below it, the `maxresp` is applied to each of the lowest level contexts above the the actual peer contexts at a time. For example, if the `GrandParent` context is `FS` (file systems) and the system has three volume groups, then a `maxresp` value of 2 could cause up to a maximum of 2 x 3 = 6 responses to be generated.
threshold—Must be non-zero if `excp_type` specifies that exceptions or SNMP traps must be generated. If specified as zero, indicates that all values read qualify to be returned in feeds. The value specified is compared to the data value read for each peer statistic. If the data value exceeds the `threshold`, it qualifies to be returned as an `SpmiHotItems` element in the `SpmiHotVals` structure. If the `threshold` is specified as a negative value, the value qualifies if it is lower than the numeric value of `threshold`. If `feed_type` is specified as `SiHotAlways`, the threshold value is ignored for feeds. For peer statistics of type `SiCounter`, the `threshold` must be specified as a rate per second; for `SiQuantity` statistics the `threshold` is specified as a level.

frequency—Must be non-zero if `excp_type` specifies that exceptions or SNMP traps must be generated. Ignored for feeds. Specifies the minimum number of minutes that must expire between any two exceptions/traps generated from this `SpmiHotVals` structure. This value must be specified as no less than 5 minutes.

feed_type—Specifies if feeds of `SpmiHotItems` should be returned for this `SpmiHotVals` structure. The following values are valid:
- `SiHotNoFeed`—No feeds should be generated.
- `SiHotThreshold`—Feeds are controlled by `threshold`.
- `SiHotAlways`—All values, up-to a maximum of `maxresp` must be returned as feeds.

excp_type—Controls the generation of exception data packets and/or the generation of SNMP Traps from `xmservd`. Note that these types of packets and traps can only actually be sent if `xmservd` is running. Because of this, exception packets and SNMP traps are only generated as long as `xmservd` is active. Traps can only be generated on AIX. The conditions for generating exceptions and traps are controlled by the `threshold` and `frequency` parameters. The following values are valid for `excp_type`:
- `SiNoHotException`—Generate neither exceptions nor traps.
- `SiHotException`—Generate exceptions but not traps.
- `SiHotTrap`—Generate SNMP traps but not exceptions.
- `SiHotBoth`—Generate both exceptions and SNMP traps.

severity—Required to be positive and greater than zero if exceptions are generated, otherwise specify as zero. Used to assign a severity code to the exception for display by `exmon`.

trap_no—Required to be positive and greater than zero if SNMP traps are generated, otherwise specify as zero. Used to assign the trap number in the generated SNMP trap.

This subroutine is part of the Performance Toolbox for AIX licensed product.

**Return Values**
If successful, the subroutine returns a pointer to a structure of type `struct SpmiHotVals`. If an error occurs, NULL is returned and an error text may be placed in the external character array `RSiEMsg`. If you attempt to add more values to a statset than the current local buffer size allows, `RSiErrno` is set to `RSITooMany`. If you attempt to add more values than the buffer size of the remote host’s `xmservd` daemon allows, `RSiErrno` is set to `RSiBadStat` and the status field in the returned packet is set to `too_many_values`.

The external integer `RSiMaxValues` holds the maximum number of values acceptable with the data-consumer’s buffer size.

**Error Codes**
All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:
- extern char `RSiEMsg[]`;
• extern int RSiErrno;

If the subroutine returns without an error, the RSiErrno variable is set to RSiOkay and the RSiEMsg character array is empty. If an error is detected, the RSiErrno variable returns an error code, as defined in the enum RSiErrorType. RSi error codes are described in List of RSi Error Codes.

Files

/usr/include/sys/Rsi.h Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

Related Information

For related information, see:
• “RSiCreateHotSet Subroutine” on page 71
• “RSiOpen Subroutine” on page 90.

RSiChangeFeed Subroutine

Purpose
Changes the frequency at which the xmservd on the host identified by the first argument daemon is sending data_feed packets for a statset.

Library
RSI Library (libSpmi.a)

Syntax

#include sys/Rsi.h
int RSiChangeFeed(rhandle, statset, msecs)
RSiHandle rhandle; struct SpmiStatSet *statset; int msecs;

Parameters

rhandle Must be an RSiHandle, which was previously initialized by the RSiOpen (“RSiOpen Subroutine” on page 90) subroutine.

statset Must be a pointer to a structure of type struct SpmiStatSet which was previously returned by a successful RSiCreateStatSet subroutine call. Data feeding must have been started for this SpmiStatSet via a previous RSiStartFeed (“RSiStartFeed Subroutine” on page 95) subroutine call.

msecs The number of milliseconds between the sending of data_feed packets. This number is rounded to a multiple of min_remote_int milliseconds by the xmservd daemon on the remote host. This minimum interval can be modified through the -i command line interval to xmservd.

This subroutine is part of the Performance Toolbox for AIX licensed product.

Return Values

If successful, the subroutine returns zero, otherwise -1. A NULL error text is placed in the external character array RSiEMsg regardless of the subroutine’s success or failure.
Error Codes

All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char RSIEMsg[];
- extern int RSIErno;

If the subroutine returns without an error, the RSIErno variable is set to RSIOkay and the RSIEMsg character array is empty. If an error is detected, the RSIErno variable returns an error code, as defined in the enum RSIErrorType. RSI error codes are described in [List of RSI Error Codes].

Files

/usr/include/sys/Rsi.h  Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

Related Information

For related information, see:

- “RSICreateStatSet Subroutine” on page 72
- “RSIOpen Subroutine” on page 90
- “RSIStartFeed Subroutine” on page 95.

RSICreateStatSet Subroutine

Purpose

Changes the frequency at which the xmservd on the host identified by the first argument daemon is sending hot_feed packets for a statset or checking if exceptions or SNMP traps should be generated.

Library

RSI Library (libSpmi.a)

Syntax

```c
#include <sys/Rsi.h>
int RSICreateHotFeed(rhandle, hotset, msecs)
RSIHandle rhandle; struct SpmiHotSet *hotset; int msecs;
```

Parameters

- `rhandle` Must be an RSIHandle, which was previously initialized by the RSIOpen ("RSIOpen Subroutine" on page 90) subroutine.
- `hotset` Must be a pointer to a structure of type struct SpmiHotSet, which was previously returned by a successful RSICreateHotSet ("RSICreateHotSet Subroutine" on page 71) subroutine call. Data feeding must have been started for this SpmiHotSet via a previous RSIStartHotFeed ("RSIStartHotFeed Subroutine" on page 96) subroutine call.
- `msecs` The number of milliseconds between the sending of Hot_feed packets. This number is rounded to a multiple of min_remote_int milliseconds by the xmservd daemon on the remote host. This minimum interval can be modified through the -i command line interval to xmservd.

This subroutine is part of the Performance Toolbox for AIX licensed product.
Return Values
If successful, the subroutine returns zero, otherwise -1. A NULL error text is placed in the external character array \texttt{RSiEMsg} regardless of the subroutine’s success or failure.

Error Codes
All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:
\begin{itemize}
    \item \texttt{extern char RSiEMsg[];}
    \item \texttt{extern int RSiErrno;}
\end{itemize}

If the subroutine returns without an error, the \texttt{RSiErrno} variable is set to \texttt{RSiOkay} and the \texttt{RSiEMsg} character array is empty. If an error is detected, the \texttt{RSiErrno} variable returns an error code, as defined in the enum \texttt{RSiErrorType}. RSi error codes are described in \hyperref[RSiErrorCodes]{List of RSi Error Codes}.

Files
\texttt{/usr/include/sys/Rsi.h} Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

Related Information
In the sample program, the \texttt{SpmiStatSet} is created in the local function \texttt{lststats} shown previously in lines 6 through 10.
\begin{itemize}
    \item \hyperref[RSiCreateHotSetSubroutine]{“RSiCreateHotSet Subroutine” on page 71}
    \item \hyperref[RSiOpenSubroutine]{“RSiOpen Subroutine” on page 90}
    \item \hyperref[RSiStartHotFeedSubroutine]{“RSiStartHotFeed Subroutine” on page 96}
\end{itemize}

\textbf{RSiClose Subroutine}

Purpose
Terminates the RSI interface for a remote host connection.

Library
RSI Library (libSpmi.a)

Syntax
\begin{verbatim}
#include sys/Rsi.h
void RSiClose(rhandle)
    RSiHandle rhandle;
\end{verbatim}

Description
The \texttt{RSiClose} subroutine is responsible for:
\begin{enumerate}
    \item Removing the data-consumer program as a known data consumer on a particular host. This is done by sending a \texttt{going_down} packet to the host.
    \item Marking the RSI handle as not active.
    \item Releasing all memory allocated in connection with the RSI handle.
    \item Terminating the RSI interface for a remote host.
\end{enumerate}

A successful \hyperref[RSiOpenSubroutine]{RSiOpen} subroutine creates tables on the remote host it was issued against. Therefore, a data consumer program that has issued successful \hyperref[RSiOpenSubroutine]{RSiOpen}
subroutine calls should issue an **RSiClose** subroutine call for each RSiOpen call before the program exits so that the tables in the remote **xmservd** daemon can be released.

This subroutine is part of the Performance Toolbox for AIX licensed product.

**Parameters**

**rhandle**Must be an **RSiHandle**, which was previously initialized by the **RSiOpen** subroutine.

The macro **RSiIsOpen** can be used to test whether an RSI handle is open. It takes an **RSiHandle** as argument and returns true (1) if the handle is open, otherwise false (0).

**Files**

/usr/include/sys/Rsi.hDeclares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

**Related Information**

For related information, see:

- **“RSiInit Subroutine” on page 83**
- **“RSiOpen Subroutine” on page 90**

---

**RSiCreateHotSet Subroutine**

**Purpose**

Creates an empty hotset on the remote host identified by the argument.

**Library**

RSI Library (libSpmi.a)

**Syntax**

```c
#include sys/Rsi.h
struct SpmiHotSet *RSiCreateHotSet(rhandle)
RSiHandle rhandle;
```

**Description**

The **RSiCreateHotSet** subroutine allocates an **SpmiHotSet** structure. The structure is initialized as an empty **SpmiHotSet** and a pointer to the **SpmiHotSet** structure is returned.

The **SpmiHotSet** structure provides the anchor point to a set of peer statistics and must exist before the **RSiAddSetHot** subroutine can be successfully called.

This subroutine is part of the Performance Toolbox for AIX licensed product.

**Parameters**

**rhandle**Must be an **RSiHandle**, which was previously initialized by the **RSiOpen** subroutine.

**Return Values**

The **RSiCreateHotSet** subroutine returns a pointer to a structure of type **SpmiHotSet** if successful. If unsuccessful, the subroutine returns a NULL value.
Error Codes
All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:
- extern char RSiEMsg[];
- extern int RSiErrno;

If the subroutine returns without an error, the RSiErrno variable is set to RSiOkay and the RSiEMsg character array is empty. If an error is detected, the RSiErrno variable returns an error code, as defined in the enum RSiErrorType. RSI error codes are described in List of RSI Error Codes.

Files
/usr/include/sys/Rsi.h Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

Related Information
For related information, see:
- "RSiOpen Subroutine" on page 90
- "RSiAddSetHot Subroutine" on page 65.

RSiCreateStatSet Subroutine

Purpose
Creates an empty statset on the remote host identified by the argument.

Library
RSI Library (libSpmi.a)

Syntax
#include sys/Rsi.h
struct SpmiStatSet *RSiCreateStatSet(rhandle)
RSiHandle rhandle;

Description
The RSiCreateStatSet subroutine allocates an SpmiStatSet structure. The structure is initialized as an empty SpmiStatSet and a pointer to the SpmiStatSet structure is returned.

The SpmiStatSet structure provides the anchor point to a set of statistics and must exist before the RSiPathAddSetStat ("RSiPathAddSetStat Subroutine" on page 93) subroutine can be successfully called.

This subroutine is part of the Performance Toolbox for AIX licensed product.

Parameters
rhandleMust be an RSIHandle, which was previously initialized by the RSiOpen ("RSiOpen Subroutine" on page 90) subroutine.
Return Values
The RSICreateStatSet subroutine returns a pointer to a structure of type SpmiStatSet if successful. If unsuccessful, the subroutine returns a NULL value.

Error Codes
All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:
- extern char RSiEMsg[];
- extern int RSiErrno;

If the subroutine returns without an error, the RSiErrno variable is set to RSiOkay and the RSiEMsg character array is empty. If an error is detected, the RSiErrno variable returns an error code, as defined in the enum RSiErrorType. RSi error codes are described in [List of RSI Error Codes].

Files
/usr/include/sys/Rsi.h Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

Related Information
For related information, see:
- “RSIOpen Subroutine” on page 90
- “RSIPathAddSetStat Subroutine” on page 93.

RSiDelSetHot Subroutine

Purpose
Deletes a single set of peer statistics identified by an SpmiHotVals structure from an SpmiHotSet.

Library
RSI Library (libSpmi.a)

Syntax
#include <sys/Rsi.h>
int RSiDelSetHot(rhandle, hsp, hvp)
RSiHandle rhandle; struct SpmiHotSet *hsp; struct SpmiHotVals*hvp;

Description
The RSiDelSetHot subroutine performs the following actions:
1. Validates that the SpmiHotSet identified by the second argument exists and contains the SpmiHotVals statistic identified by the third argument.
2. Deletes the SpmiHotVals value from the SpmiHotSet so that future data_feed packets do not include the deleted statistic.

This subroutine is part of the Performance Toolbox for AIX licensed product.

Parameters
- rhandle: Must be an RSiHandle, which was previously initialized by the RSiOpen ("RSIOpen Subroutine" on page 90) subroutine.
**hsp** Must be a pointer to a structure type `struct SpmiHotSet` which was previously returned by a successful **RSiCreateHotSet** subroutine call.

**hvp** Must be a handle of type `struct SpmiHotVals` as returned by a successful **RSiAddSetHot** subroutine call. You cannot specify an `SpmiHotVals` that was internally generated by the Spmi library code as described under the **GrandParent** parameter to **RSiAddSetHot**.

**Return Values**
If successful, the subroutine returns a zero value; otherwise it returns a non-zero value and an error text may be placed in the external character array **RSiEMsg**.

**Error Codes**
All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- `extern char RSiEMsg[];`
- `extern int RSiErrno;`

If the subroutine returns without an error, the **RSiErrno** variable is set to **RSiOkay** and the **RSiEMsg** character array is empty. If an error is detected, the **RSiErrno** variable returns an error code, as defined in the enum **RSiErrorType**. RSi error codes are described in [List of RSi Error Codes](#).

**Files**

```
/usr/include/sys/Rsi.h
```
Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

**Related Information**
For related information, see:

- [“RSiOpen Subroutine” on page 90](#)
- [“RSiAddSetHot Subroutine” on page 65](#)

---

**RSiDelSetStat Subroutine**

**Purpose**
Deletes a single statistic identified by an `SpmiStatVals` pointer from an `SpmiStatSet`.

**Library**
RSI Library (libSpmi.a)

**Syntax**

```
#include <sys/Rsi.h>

int RSiDelSetStat(rhandle, ssp, svp)
RSiHandle rhandle; struct SpmiStatSet *ssp; struct SpmiStatVals *svp;
```

**Description**
The **RSiDelSetStat** subroutine performs the following actions:

1. Validates the `SpmiStatSet` identified by the second argument exists and contains the `SpmiStatVals` statistic identified by the third argument.
2. Deletes the SpmiStatVals value from the SpmiStatSet so that future data_feed packets do not include the deleted statistic.

This subroutine is part of the Performance Toolbox for AIX licensed product.

**Parameters**

**rhandle** Must be an RSiHandle, which was previously initialized by the RSiOpen ("RSiOpen Subroutine" on page 90) subroutine.

**ssp** Must be a pointer to a structure type struct SpmiStatSet, which was previously returned by a successful RSiCreateStatSet ("RSiCreateStatSet Subroutine" on page 72) subroutine call.

**svp** Must be a handle of type struct SpmiStatVals as returned by a successful RSiPathAddSetStat ("RSiPathAddSetStat Subroutine" on page 93) subroutine call.

**Return Values**

If successful, the subroutine returns a zero value; otherwise it returns a non-zero value and an error text may be placed in the external character array RSiEMsg.

**Error Codes**

All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char RSiEMsg[];
- extern int RSiErrno;

If the subroutine returns without an error, the RSiErrno variable is set to RSiOkay and the RSiEMsg character array is empty. If an error is detected, the RSiErrno variable returns an error code, as defined in the enum RSiErrorType. RSi error codes are described in [List of RSi Error Codes](#).

**Files**

/usr/include/sys/Rsi.h Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

**Related Information**

For related information, see:

- "RSiCreateStatSet Subroutine" on page 72
- "RSiOpen Subroutine" on page 90
- "RSiPathAddSetStat Subroutine" on page 93.

**RSiFirstCx Subroutine**

**Purpose**

Returns the first subcontext of an SpmiCx context.

**Library**

RSI Library (libSpmi.a)

**Syntax**

#include sys/Rsi.h
struct SpmiCxLink *RSiFirstCx(rhandle, context, name, descr)

RSiHandle rhandle;
ctx_handle *context;
char **name;
char **descr;

**Description**

The `RSiFirstCx` subroutine performs the following actions:
1. Validates that the context identified by the second argument exists.
2. Returns a handle to the first element of the list of subcontexts defined for the context.
3. Returns the short name and description of the subcontext.

This subroutine is part of the Performance Toolbox for AIX licensed product.

**Parameters**

- **rhandle** Must be an `RSiHandle`, which was previously initialized by the `RSiOpen` subroutine.

- **context** Must be a handle of type `ctx_handle`, which was previously returned by a successful `RSiPathGetCx` subroutine call.

- **name** Must be a pointer to a pointer to a character array. The pointer must be initialized to point at a character array pointer. When the subroutine call is successful, the short name of the subcontext is returned in the character array pointer.

- **descr** Must be a pointer to a pointer to a character array. The pointer must be initialized to point at a character array pointer. When the subroutine call is successful, the description of the subcontext is returned in the character array pointer.

**Return Values**

If successful, the subroutine returns a pointer to a structure of type `struct SpmiCxLink`. If an error occurs or if the context doesn't contain subcontexts, NULL is returned and an error text may be placed in the external character array `RSiEMsg`.

**Error Codes**

All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char `RSiEMsg`[];
- extern int `RSiErrno`;

If the subroutine returns without an error, the `RSiErrno` variable is set to `RSiOkay` and the `RSiEMsg` character array is empty. If an error is detected, the `RSiErrno` variable returns an error code, as defined in the enum `RSiErrorType`. RSI error codes are described in the List of RSI Error Codes.

**Files**

/usr/include/sys/Rsi.h Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.
Related Information
For related information, see:
• “RSiNextCx Subroutine” on page 88
• “RSiOpen Subroutine” on page 90
• “RSiPathGetCx Subroutine” on page 94.

RSiFirstStat Subroutine

Purpose
Returns the first statistic of an SpmiCx context.

Library
RSI Library (libSpmi.a)

Syntax
#include sys/Rsi.h
struct SpmiStatLink *RSiFirstStat(rhandle, context, name, descr)
rhandle;
context;
name;
descr;

Description
The RSiFirstStat subroutine performs the following actions:
1. Validates that the context identified by the second argument exists.
2. Returns a handle to the first element of the list of statistics defined for the context.
3. Returns the short name and description of the statistic.

This subroutine is part of the Performance Toolbox for AIX licensed product.

Parameters
rhandle Must be an RSiHandle, which was previously initialized by the RSiOpen subroutine.
context Must be a handle of type cx_handle, which was previously returned by a successful subroutine call.
name Must be a pointer to a pointer to a character array. The pointer must be initialized to point at a character array pointer. When the subroutine call is successful, the short name of the statistics value is returned in the character array pointer.

Return Values
If successful, the subroutine returns a pointer to a structure of type struct SpmiStatLink. If an error occurs, NULL is returned and an error text may be placed in the external character array RSiEMsg.
Error Codes

All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char RSiEMsg[];
- extern int RSiErrno;

If the subroutine returns without an error, the RSiErrno variable is set to RSiOkay and the RSiEMsg character array is empty. If an error is detected, the RSiErrno variable returns an error code, as defined in the enum RSiErrorType. RSi error codes are described in [List of RSi Error Codes](#).

Files

/usr/include/sys/Rsi.h

Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

Related Information
For related information, see:

- "RSiNextStat Subroutine" on page 89
- "RSiOpen Subroutine" on page 90
- "RSiPathGetCx Subroutine" on page 94.

RSiGetHotItem Subroutine

Purpose
Locates and decodes the next SpmiHotItems element at the current position in an incoming data packet of type hot_feed.

Library
RSI Library (libSpmi.a)

Syntax

```c
#include sys/Rsi.h

struct SpmiHotVals *RSiGetHotItem(rhandle, HotSet, index, value, absvalue, name)
RSiHandle rhandle;
struct SpmiHotSet **HotSet;
int *index;
float *value;
float absvalue;
char **name;
```

Description
The RSiGetHotItem subroutine locates the SpmiHotItems structure in the hot_feed data packet indexed by the value of the index parameter. The subroutine returns a NULL value if no further SpmiHotItems structures are found. The RSiGetHotItem subroutine should only be executed after a successful call to the RSiGetHotSet subroutine.

The RSiGetHotItem subroutine is designed to be used for walking all SpmiHotItems elements returned in a hot_feed data packet. Because the data packet may contain elements belonging to more than one SpmiHotSet, the index is purely abstract and is only used to keep position. By feeding the updated integer pointed to by index back to the next call, the walking of the hot_feed packet can be done in a tight loop.
Successful calls to \texttt{RSiGetHotItem} will decode each \texttt{SpmiHotItems} element and return the data value in \texttt{value} and the name of the peer context that owns the corresponding statistic in \texttt{name}.

This subroutine is part of the Performance Toolbox for AIX licensed product.

**Parameters**

\begin{itemize}
\item \textbf{rhandle} Must be an \texttt{RSiHandle}, which was previously initialized by the \texttt{RSiOpen} subroutine.
\item \textbf{HotSet} Used to return a pointer to a valid \texttt{SpmiHotSet} structure as obtained by a previous \texttt{RSiCreateHotSet} subroutine call. The calling program can use this value to locate the \texttt{SpmiHotSet} if its address was stored by the program after it was created. The time stamps in the \texttt{SpmiHotSet} are updated with the time stamps of the decoded \texttt{SpmiHotItems} element.
\item \textbf{index} A pointer to an integer that contains the desired relative element number in the \texttt{SpmiHotItems} array across all \texttt{SpmiStatVals} contained in the data packet. A value of zero points to the first element. When the \texttt{RSiGetHotItem} subroutine returns, the integer contain the index of the next \texttt{SpmiHotItems} element in the data packet. By passing the returned \texttt{index} parameter to the next call to \texttt{RSiGetHotItem}, the calling program can iterate through all \texttt{SpmiHotItems} elements in the hot feed data packet.
\item \textbf{value} A pointer to a float variable. A successful call will return the decoded data value of the peer statistic. Before the value is returned, the \texttt{RSiGetHotItem} function:
\begin{itemize}
\item Determines the format of the data field as being either \texttt{SiFloat} or \texttt{SiLong} and extracts the data value for further processing.
\item Determines the data value as being either type \texttt{SiQuantity} or type \texttt{SiCounter} and performs one of the actions listed here:
\begin{itemize}
\item If the data value is of type \texttt{SiQuantity}, the subroutine returns the \texttt{val} field of the \texttt{SpmiHotItems} structure.
\item If the data value is of type \texttt{SiCounter}, the subroutine returns the value of the \texttt{val change} field of the \texttt{SpmiHotItems} structure divided by the elapsed number of seconds since the previous time a data value was requested for this set of statistics.
\end{itemize}
\end{itemize}
\item \textbf{absvalue} A pointer to a float variable. A successful call will return the decoded value of the \texttt{val} field of the \texttt{SpmiHotItems} structure of the peer statistic. In case of a statistic of type \texttt{SiQuantity}, this value will be the same as the one returned in the argument \texttt{value}. In case of a peer statistic of type \texttt{SiCounter}, the value returned is the absolute value of the counter.
\item \textbf{name} A pointer to a character pointer. A successful call will return a pointer to the name of the peer context for which the data value was read.
\end{itemize}

**Return Values**

The \texttt{RSiGetHotItem} subroutine returns a pointer to the current \texttt{SpmiHotVals} structure within the hotset. If no more \texttt{SpmiHotItems} elements are available, the subroutine returns a NULL value. The structure returned contains the data, such as threshold, which may be relevant for presentation of the results of an \texttt{SpmiGetHotSet} subroutine call to end-users. In the returned \texttt{SpmiHotVals} structure, all fields contain the correct values as declared, except for the following:

\begin{itemize}
\item \textbf{stat} Declared as \texttt{SpmiStatHdl}, actually points to a valid \texttt{SpmiStat} structure. By casting the handle to a pointer to \texttt{SpmiStat}, data in the structure can be accessed.
\item \textbf{grandpa} Contains the \texttt{cx handle} for the parent context of the peer contexts.
\item \textbf{items} When using the \texttt{Spmi} interface this is an array of \texttt{SpmiHotItems} structures. When using the \texttt{RSiGetHotItem} subroutine, the array is empty and attempts to access it will likely result in segmentation faults or access of not valid data.
\end{itemize}
**path** Will contain the path to the parent of the peer contexts. Even when the peer contexts are multiple levels below the parent context, the path points to the top context because the peer context identifiers in the SpmiHottItems elements will contain the path name from there and on. For example, if the hotvals peer set defines all volume groups, the path specified in the returned SpmiHotVals structure would be “FS” and the path name in one SpmiHottItems element may be “rootvg/lv01”. When combined with the metric name from the stat field, the full path name can be constructed as, for example, “FS/rootvg/lv01/%totfree”.

**Error Codes**
All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char RSiEMsg[];
- extern int RSiErrno;

If the subroutine returns without an error, the RSiErrno variable is set to RSiOkay and the RSiEMsg character array is empty. If an error is detected, the RSiErrno variable returns an error code, as defined in the enum RSiErrorType. RSi error codes are described in the [List of RSi Error Codes](#).

**Files**
/usr/include/sys/Rsi.h Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

**Related Information**
For related information, see:

- “RSiOpen Subroutine” on page 90
- “RSiCreateHotSet Subroutine” on page 71.

**RSiGetRawValue Subroutine**

**Purpose**
Returns a pointer to a valid SpmiStatVals structure for a given SpmiStatVals pointer by extraction from a data_feed packet. This subroutine call should only be issued from a callback function after it has been verified that a data_feed packet was received from the host identified by the first argument.

**Library**
RSI Library (libSpmi.a)

**Syntax**
```
#include <sys/Rsi.h>
struct SpmiStatVals RSiGetRawValue(rhandle, svp, index)
RSiHandle rhandle;
struct SpmiStatVals *svp;
int *index;
```

**Description**
The RSiGetRawValue subroutine performs the following:
1. Finds an SpmiStatVals structure in the received data packet based upon the second argument to the subroutine call. This involves a lookup operation in tables maintained internally by the RSi interface.
2. Updates the `struct SpmiStat` pointer in the `SpmiStatVals` structure to point at a valid `SpmiStat` structure.

3. Returns a pointer to the `SpmiStatVals` structure. The returned pointer points to a static area and is only valid until the next execution of `RSiGetRawValue`.

4. Updates an integer variable with the index into the `ValsSet` array of the `data_feed` packet, which corresponds to the second argument to the call.

This subroutine is part of the Performance Toolbox for AIX licensed product.

**Parameters**

- `rhandle` Must be an `RSiHandle`, which was previously initialized by the `RSiOpen` subroutine.
- `svp` A handle of type `struct SpmiStatVals`, which was previously returned by a successful `RSiPathAddSetStat` subroutine call.
- `index` A pointer to an integer variable. When the subroutine call succeeds, the index into the `ValsSet` array of the data feed packet is returned. The index corresponds to the element that matches the `svp` argument to the subroutine.

**Return Values**

If successful, the subroutine returns a pointer; otherwise NULL is returned and an error text may be placed in the external character array `RSiEMsg`.

**Error Codes**

All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- `extern char RSiEMsg[];
- extern int RSiErrno;

If the subroutine returns without an error, the `RSiErrno` variable is set to `RSiOkay` and the `RSiEMsg` character array is empty. If an error is detected, the `RSiErrno` variable returns an error code, as defined in the enum `RSiErrorType`. RSI error codes are described in `List of RSI Error Codes`.

**Files**

`/usr/include/sys/Rsi.h` Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

**Related Information**

For related information, see:

- "RSiOpen Subroutine" on page 90
- "RSiPathAddSetStat Subroutine" on page 93.

**RSiGetValue Subroutine**

**Purpose**

Returns a data value for a given `SpmiStatVals` pointer by extraction from the `data_feed` packet. This subroutine call should only be issued from a callback function after it has been verified that a `data_feed` packet was received from the host identified by the first argument.
Library
RSI Library (libSpmi.a)

Syntax
#include sys/Rsi.h
float RSiGetValue(rhandle,svp)
RSiHandle rhandle;
struct SpmiStatVals *svp;

Description
The RSiGetValue subroutine provides the following:
1. Finds an SpmiStatVals structure in the received data packet based upon the second argument to the subroutine call. This involves a lookup operation in tables maintained internally by the RSi interface.
2. Determines the format of the data field as being either SiFloat or SiLong and extracts the data value for further processing based upon its data format.
3. Determines the value as either of type SiQuantity or SiCounter. If the former is the case, the data value returned is the val field in the SpmiStatVals structure. If the latter type is found, the value returned by the subroutine is the val_change field divided by the elapsed number of seconds since the previous data packet's time stamp.

This subroutine is part of the Performance Toolbox for AIX licensed product.

Parameters
rhandle Must be an RSiHandle, previously initialized by the RSiOpen subroutine.

svp A handle of type struct SpmiStatVals which was previously returned by a successful RSiPathAddSetStat subroutine call.

Return Values
If successful, the subroutine returns a non-negative value; otherwise it returns a negative value less than or equal to -1.0. A NULL error text is placed in the external character array RSiErrMsg regardless of the subroutine’s success or failure.

Error Codes
All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

• extern char RSiErrMsg[];
• extern int RSiErrno;

If the subroutine returns without an error, the RSiErrno variable is set to RSiOkay and the RSiErrMsg character array is empty. If an error is detected, the RSiErrno variable returns an error code, as defined in the enum RSiErrorException. RSi error codes are described in List of RSi Error Codes.

Files
/usr/include/sys/Rsi.h Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.
Related Information

For related information, see:
- “RSiOpen Subroutine” on page 90
- “RSiPathAddSetStat Subroutine” on page 93

RSiInit Subroutine

Purpose
Allocates or changes the table of RSi handles.

Library
RSI Library (libSpmi.a)

Syntax
```c
#include <sys/Rsi.h>

RSiHandle RSiInit(count)
int count;
```

Description
Before any other RSi call is executed, a data-consumer program must issue the RSiInit call. Its purpose is to either:
- Allocate an array of RSiHandleStruct structures and return the address of the array to the data-consumer program.
- Increase the size of a previously allocated array of RSiHandleStruct structures and initialize the new array with the contents of the previous one.

This subroutine is part of the Performance Toolbox for AIX licensed product.

Parameters
count
Must specify the number of elements in the array of RSi handles. If the call is used to expand a previously allocated array, this argument must be larger than the current number of array elements. It must always be larger than zero. Specify the size of the array to be at least as large as the number of hosts your data-consumer program can talk to at any point in time.

Return Values
If successful, the subroutine returns the address of the allocated array. If an error occurs, an error text is placed in the external character array RSIEMsg and the subroutine returns NULL. When used to increase the size of a previously allocated array, the subroutine first allocates the new array, then moves the entire old array to the new area. Application programs should, therefore, refer to elements in the RSI handle array by index rather than by address if they anticipate the need for expanding the array. The array only needs to be expanded if the number of remote hosts a data-consumer program talks to might increase over the life of the program.

An application that calls RSiInit repeatedly needs to preserve the previous address of the RSiHandle array while the RSiInit call is re-executed. After the call has completed successfully, the calling program should free the previous array using the free subroutine.

Error Codes
All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:
If the subroutine returns without an error, the `RSiErrno` variable is set to `RSiOkay` and the `RSiEMsg` character array is empty. If an error is detected, the `RSiErrno` variable returns an error code, as defined in the enum `RSiErrorType`. RSi error codes are described in the list of RSi Error Codes.

**Files**

```
/usr/include/sys/Rsi.h
```

Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

**Related Information**

For related information, see the "RSiClose Subroutine" on page 70.

---

**RSiInstantiate Subroutine**

**Purpose**

Creates (instantiates) all subcontexts of an `SpmiCx` context object.

**Library**

RSI Library (`libSpmi.a`)

**Syntax**

```
#include sys/Rsi.h

int RSiInstantiate(rhandle, context)
RSiHandle rhandle;
ctx_handle *context;
```

**Description**

The `RSiInstantiate` subroutine performs the following actions:

1. Validates that the context identified by the second argument exists.
2. Instantiates the context so that all subcontexts of that context are created in the context hierarchy.
   
   Note that this subroutine call currently only makes sense if the context's `SiInstFreq` is set to `SiContInst` or `SiCfgInst` because all other contexts would have been instantiated whenever the `xmservd` daemon was started.

The `RSiInstantiate` subroutine explicitly instantiates the subcontexts of an instantiable context. If the context is not instantiable, do not call the `RSiInstantiate` subroutine.

This subroutine is part of the Performance Toolbox for AIX licensed product.

**Parameters**

- `rhandle` Must point to a structure of type `RSiHandle`, which was previously initialized by the `RSiOpen` subroutine.

- `context` Must be a handle of type `cx_handle`, which was previously returned by a successful `RSiPathGetCx` subroutine call.
Return Values
If successful, the subroutine returns a zero value; otherwise it returns an error code as defined in SiError and an error text may be placed in the external character array RSiEMsg.

Error Codes
All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char RSiEMsg[];
- extern int RSiErrno;

If the subroutine returns without an error, the RSiErrno variable is set to RSiOkay and the RSiEMsg character array is empty. If an error is detected, the RSiErrno variable returns an error code, as defined in the enum RSiErrorType. RSi error codes are described in List of RSI Error Codes.

Files
/usr/include/sys/Rsi.h Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

Related Information
For related information, see:
- "RSiFirstCx Subroutine" on page 75
- "RSiOpen Subroutine" on page 90
- "RSiPathGetCx Subroutine" on page 94.

RSiInvite Subroutine

Purpose
Invites data suppliers on the network to identify themselves and returns a table of data-supplier host names.

Library
RSI Library (libSpmi.a)

Syntax
```
#include sys/Rsi.h

char **RSiInvite(resy_callb, excp_callb)
int (*resy_callb)();
int (*excp_callb)();
```

Description
The RSiInvite subroutine call broadcasts are_you_there messages on the network to provoke xmservd daemons on remote hosts to respond and returns a table of all responding hosts.

This subroutine is part of the Performance Toolbox for AIX licensed product.

Parameters
The arguments to the subroutine are:
resy_callb: Must be either NULL or a pointer to a function that processes i_am_back packets as they are received from the xmservd daemons on remote hosts for the duration of the RSiInvite subroutine call. When the callback function is invoked, it is passed three arguments as described in the following information.

If this argument is specified as NULL, a callback function internal to the RSiInvite subroutine receives any i_am_back packets and uses them to build the table of host names the function returns.

excp_callb: Must be NULL or a pointer to a function that processes except_rec packets as they are received from the xmservd daemons on remote hosts. If a NULL pointer is passed, your application does not receive except_rec messages. When this callback function is invoked, it is passed three arguments as described in the following information.

This argument always overrides the corresponding argument of any previous RSiInvite or RSiOpen call, and it can be overridden by subsequent executions of either. In this way, your application can turn exception monitoring on and off. For an RSiOpen to override the exception processing specified by a previous open call, the connection must first be closed with the RSiClose call. That’s because an RSiOpen against an already active handle is treated as a no-operation.

The resy_callb and excp_callb functions in your application are called with the following three arguments:

- An RSiHandle. The RSi handle pointed to is almost certain not to represent the host that sent the packet. Ignore this argument, and use only the second one: the pointer to the input buffer.
- A pointer of type pack * to the input buffer containing the received packet. Always use this pointer rather than the pointer in the RSiHandle structure.
- A pointer of type struct sockaddr_in * to the IP address of the originating host.

Return Values

If successful, the subroutine returns an array of character pointers, each of which contains a host name of a host that responded to the invitation. The returned host names are actually constructed as two “words” with the first one being the host name returned by the host in response to an are_you_there request; the second one being the character form of the host’s IP address. The two “words” are separated by one or more blanks. This format is suitable as an argument to the RSiOpen (“RSiOpen Subroutine” on page 90) subroutine call. In addition, the external integer variable RSiInvTabActive contains the number of host names found. The returned pointer to an array of host names must not be freed by the subroutine call. The calling program should not assume that the pointer returned by this subroutine call remains valid after subsequent calls to RSiInvite. If the call is not successful, an error text is placed in the external character array RSiEMsg, an error number is placed in RSiErrno, and the subroutine returns NULL.

The list of host names returned by RSiInvite does not include the hosts your program has already established a connection with through an RSiOpen call. Your program is responsible for keeping track of such hosts. If you need a list of both sets of hosts, either let the RSiInvite call be the first one issued from your program or merge the list of host names returned by the call with the list of hosts to which you have connections.

Error Codes

All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char RSiEMsg[];
- extern int RSiErrno;

If the subroutine returns without an error, the RSiErrno variable is set to RSIOkay and the RSiEMsg character array is empty. If an error is detected, the RSiErrno variable returns an error code, as defined in the enum RSIErrorType. RSI error codes are described in List of RSI Error Codes.
Files

/usr/include/sys/Rsi.h Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

Related Information
For related information, see "RSiOpen Subroutine" on page 90.

RSiMainLoop Subroutine

Purpose
Allows an application to suspend execution and wait to get awakened when data feeds arrive.

Library
RSI Library (libSpmi.a)

Syntax
#include sys/Rsi.h
void RSiMainLoop(msecs)
int msecs;

Description
The RSiMainLoop subroutine:
1. Allows the data-consumer program to suspend processing while waiting for data_feed packets to arrive from one or more xmservd daemons.
2. Tells the subroutine that waits for data feeds to return control to the data-consumer program so that the latter can check for and react to other events.
3. Invokes the subroutine to process data_feed packets for each such packet received.

To work properly, the RSiMainLoop subroutine requires that at least one RSiOpen ("RSiOpen Subroutine" on page 90) call has been successfully completed and that the connection has not been closed.

This subroutine is part of the Performance Toolbox for AIX licensed product.

Parameters
msecs The minimum elapsed time in milliseconds that the subroutine should continue to attempt receives before returning to the caller. Notice that your program releases control for as many milliseconds you specify but that the callback functions defined on the RSiOpen call may be called repetitively during that time.

Error Codes
All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char RSiEMsg[];
- extern int RSiErrno;

If the subroutine returns without an error, the RSiErrno variable is set to RSiOkay and the RSiEMsg character array is empty. If an error is detected, the RSiErrno variable returns an error code, as defined in the enum RSiErrorType. RSI error codes are described in List of RSI Error Codes.
Files
/usr/include/sys/Rsi.h  Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

Related Information
For related information, see "RSiOpen Subroutine" on page 90.

RSiNextCx Subroutine

Purpose
Returns the next subcontext of an SpmiCx context.

Library
RSI Library (libSpmi.a)

Syntax
#include <sys/Rsi.h>
struct SpmiCxLink *RSiNextCx(rhandle, context, link, name, descr)
RSiHandle rhandle;
cx_handle *context;
struct SpmiCxLink *link;
char **name;
char **descr;

Description
The RSiNextCx subroutine:
1. Validates that the context identified by the second argument exists.
2. Returns a handle to the next element of the list of subcontexts defined for the context.
3. Returns the short name and description of the subcontext.

This subroutine is part of the Performance Toolbox for AIX licensed product.

Parameters
rhandle Must point to a structure of type RSiHandle, which was previously initialized by the RSiOpen ("RSiOpen Subroutine" on page 90) subroutine.

cx_handle Must be a handle of type cx_handle, which was previously returned by a successful RSiPathGetCx ("RSiPathGetCx Subroutine" on page 94) subroutine call.

link Must be a pointer to a structure of type struct SpmiCxLink, which was previously returned by a successful RSiFirstCx ("RSiFirstCx Subroutine" on page 75) or RSiNextCx subroutine call.

name Must be a pointer to a pointer to a character array. The pointer must be initialized to point at a character array pointer. When the subroutine call is successful, the short name of the subcontext is returned in the character array pointer.

descr Must be a pointer to a pointer to a character array. The pointer must be initialized to point at a character array pointer. When the subroutine call is successful, the description of the subcontext is returned in the character array pointer.
Return Values
If successful, the subroutine returns a pointer to a structure of type `struct SpmiCxLink`. If an error occurs, or if no more subcontexts exist for the context, NULL is returned and an error text may be placed in the external character array `RSiEMsg`.

Error Codes
All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char RSiEMsg[];
- extern int RSiErrno;

If the subroutine returns without an error, the `RSiErrno` variable is set to `RSiOkay` and the `RSiEMsg` character array is empty. If an error is detected, the `RSiErrno` variable returns an error code, as defined in the enum `RSiErrorType`. RSI error codes are described in [List of RSI Error Codes](#).

Files
`/usr/include/sys/Rsi.h` Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

Related Information
For related information, see:

- "RSiFirstChildCx Subroutine" on page 75
- "RSiOpen Subroutine" on page 90
- "RSiPathGetCx Subroutine" on page 94.

RSiNextStat Subroutine

Purpose
Returns the next statistic of an `SpmiCx` context.

Library
RSI Library (`libSpmi.a`)

Syntax

```c
#include sys/Rsi.h
struct SpmiStatLink *RSiNextStat(rhandle, context, link, name, descr)
RSiHandle rhandle;
cx_handle *context;
struct SpmiStatLink *link;
char **name;
char **descr;
```

Description
The `RSiNextStat` subroutine:
1. Validates that a context identified by the second argument exists.
2. Returns a handle to the next element of the list of statistics defined for the context.
3. Returns the short name and description of the statistic.
This subroutine is part of the Performance Toolbox for AIX licensed product.

**Parameters**

- **rhandle**: Must be an `RSHandle`, which was previously initialized by the [RSiOpen subroutine](#).

- **context**: Must be a handle of type `cx_handle`, which was previously returned by a successful [RSiPathGetCx subroutine](#) call.

- **link**: Must be a pointer to a structure of type `struct SpmiStatLink`, which was previously returned by a successful [RSiFirstStat subroutine](#) or [RSiNextStat subroutine](#) call.

- **name**: Must be a pointer to a pointer to a character array. The pointer must be initialized to point at a character array pointer. When the subroutine call is successful, the short name of the statistics value is returned in the character array pointer.

- **descr**: Must be a pointer to a pointer to a character array. The pointer must be initialized to point at a character array pointer. When the subroutine call is successful, the description of the statistics value is returned in the character array pointer.

**Return Values**

If successful, the subroutine returns a pointer to a structure of type `struct SpmiStatLink`. If an error occurs, or if no more statistics exists for the context, NULL is returned and an error text may be placed in the external character array `RSiEMsg`.

**Error Codes**

All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- `extern char RSiEMsg[];
- extern int RSiErrno;

If the subroutine returns without an error, the `RSiErrno` variable is set to `RSiOkay` and the `RSiEMsg` character array is empty. If an error is detected, the `RSiErrno` variable returns an error code, as defined in the enum `RSiErrorType`. RSI error codes are described in [List of RSI Error Codes](#).

**Files**

```
/usr/include/sys/Rsi.h
```

Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

**Related Information**

For related information, see:
- [“RSiFirstStat Subroutine” on page 77](#)
- [“RSiOpen Subroutine”](#)
- [“RSiPathGetCx Subroutine” on page 94](#)

**RSiOpen Subroutine**

**Purpose**

Initializes the RSi interface for a remote host.
Library
RSI Library (libSpmi.a)

Syntax
#include <sys/Rsi.h>
int RSiOpen(rhandle, wait, bufsize, hostID, feed_callb, 
            resy_callb, excp_callb)
RSiHandle rhandle;
int wait;
int bufsize;
char *hostID;
int (*feed_callb)();
int (*resy_callb)();
int (*excp_callb)();

Description
The RSiOpen subroutine performs the following actions:
1. Establishes the issuing data-consumer program as a data consumer known to the xmservd daemon on a particular host. The subroutine does this by sending an are_you_there packet to the host.
2. Initializes an RSi handle for subsequent use by the data-consumer program.

This subroutine is part of the Performance Toolbox for AIX licensed product.

Parameters
The arguments to the subroutine are:

rhandle Must point to an element of the RSiHandleStruct array, which is returned by a previous RSiInit subroutine on page 83. If the subroutine is successful the structure is initialized and ready to use as a handle for subsequent RSi interface subroutine calls.

wait Must specify the timeout in milliseconds that the RSi interface shall wait for a response when using the request-response functions. On LANs, a reasonable value for this argument is 100 milliseconds. If the response is not received after the specified wait time, the library subroutines retry the receive operation until five times the wait time has elapsed before returning a timeout indication. The wait time must be zero or more milliseconds.

bufsize Specifies the maximum buffer size to be used for constructing network packets. This size must be at least 4,096 bytes. The buffer size determines the maximum packet length that can be received by your program and sets the limit for the number of data values that can be received in one data_feed packet. There’s no point in setting the buffer size larger than that of the xmservd daemon because both must be able to handle the packets. If you need large sets of values, you can use the command line argument -b of xmservd to increase its buffer size up to 16,384 bytes.

The fixed part of a data_feed packet is 104 bytes and each value takes 32 bytes. A buffer size of 4,096 bytes allows up to 124 values per packet.

hostID Must be a character array containing the identification of the remote host whose xmservd daemon is the one with which you want to talk. The first characters of the host identification (up to the first white space) is used as the host name. The full host identification is stored in the RSiHandle field longname and may contain any description that helps the end user identify the host used. The host name may be either in long format (including domain name) or in short format.

feed_callb Must be a pointer to a function that processes data_feed packets as they are received from the xmservd daemon. When this callback function is invoked, it is passed three arguments as described in the following information.
**resy_callb** Must be a pointer to a function that processes `i_am_back` packets as they are received from the `xmservd` daemon. When this callback function is invoked it is passed three arguments as described in the following information.

**excp_callb** Must be NULL or a pointer to a function that processes `except_rec` packets as they are received from the `xmservd` daemon. If a NULL pointer is passed, your application does not receive `except_rec` messages. When this callback function is invoked, it is passed three arguments as described in the following information. This argument always overrides the corresponding argument of any previous `RSiInvite` (“RSiInvite Subroutine” on page 85) or `RSiOpen` (“RSiOpen Subroutine” on page 90) subroutine call and can itself be overridden by subsequent executions of either. In this way, your application can turn exception monitoring on and off. For an `RSiOpen` call to override the exception processing specified by a previous open call, the connection must first be closed with the `RSiClose` (“RSiClose Subroutine” on page 70) subroutine call.

The `feed_callb`, `resy_callb`, and `excp_callb` functions are called with the arguments:

- **RSiHandle**. When a `data_feed` packet is received, the structure pointed to is guaranteed to represent the host sending the packet. In all other situations the `RSiHandle` structure may represent any of the hosts to which your application is talking.

- Pointer of type `pack *` to the input buffer containing the received packet. In callback functions, always use this pointer rather than the pointer in the `RSiHandle` structure.

- Pointer of type `struct sockaddr_in *` to the IP address of the originating host.

**Return Values**

If successful, the subroutine returns zero and initializes the array element of type `RSiHandle` pointed to by `rhandle`. If an error occurs, error text is placed in the external character array `RSiEMsg` and the subroutine returns a negative value.

**Error Codes**

All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char `RSiEMsg []`;
- extern int `RSiErrno`;

If the subroutine returns without an error, the `RSiErrno` variable is set to `RSiOkay` and the `RSiEMsg` character array is empty. If an error is detected, the `RSiErrno` variable returns an error code, as defined in the enum `RSiErrorType`. RSI error codes are described in the [List of RSI Error Codes](#).

**Files**

`/usr/include/sys/Rsi.h` Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

**Related Information**

For related information, see:

- “RSiClose Subroutine” on page 70
- “RSiInvite Subroutine” on page 85
RSiPathAddSetStat Subroutine

Purpose
Add a single statistics value to an already defined SpmiStatSet.

Library
RSI Library (libSpmi.a)

Syntax
```
#include <sys/Rsi.h>
struct SpmiStatVals *RSiPathAddSetStat(rhandle, statset, path)
RSiHandle rhandle;
struct SpmiStatSet *statset;
char *path;
```

Parameters
- **rhandle**: Must be an RSiHandle, which was previously initialized by the RSiOpen ("RSiOpen Subroutine" on page 90) subroutine.
- **statset**: Must be a pointer to a structure of type struct SpmiStatSet, which was previously returned by a successful RSiCreateStatSet ("RSiCreateStatSet Subroutine" on page 72) subroutine call.
- **path**: Must be the full value path name of the statistics value to add to the SpmiStatSet. The value path name must not include a terminating slash. Note that value path names never start with a slash.

Return Values
If successful, the subroutine returns a pointer to a structure of type struct SpmiStatVals. If an error occurs, NULL is returned and an error text may be placed in the external character array RSiEMsg. If you attempt to add more values to a statset than the current local buffer size allows, RSiErrno is set to RSiTooMany. If you attempt to add more values than the buffer size of the remote host's xmservd daemon allows, RSiErrno is set to RSiBadStat and the status field in the returned packet is set to too_many_values.

The external integer RSiMaxValues holds the maximum number of values acceptable with the data-consumer's buffer size.

Error Codes
All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:
- extern char RSiEMsg[];
- extern int RSiErrno;

If the subroutine returns without an error, the RSiErrno variable is set to RSiOkay and the RSiEMsg character array is empty. If an error is detected, the RSiErrno variable returns an error code, as defined in the enum RSiErrorType. Rsi error codes are described in List of RSI Error Codes.

Files
```
/usr/include/sys/Rsi.h
```
Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.
Related Information
For related information, see:
- "RSiCreateStatSet Subroutine" on page 72
- "RSiOpen Subroutine" on page 90.

RSiPathGetCx Subroutine

Purpose
Searches the context hierarchy for an SpmiCx context that matches a context path name.

Library
RSI Library (libSpmi.a)

Syntax
#include sys/Rsi.h

cx_handle *RSiPathGetCx(rhandle, path)
RSiHandle rhandle;
char *path;

Description
The RSiPathGetCx subroutine performs the following actions:
1. Searches the context hierarchy for a given path name of a context.
2. Returns a handle to be used when subsequently referencing the context.

This subroutine is part of the Performance Toolbox for AIX licensed product.

Parameters
rhandle Must be an RSiHandle, which was previously initialized by the RSiOpen ("RSiOpen Subroutine" on page 90) subroutine.

path A path name of a context for which a handle is to be returned. The context path name must be the full path name and must not include a terminating slash. Note that context path names never start with a slash.

Return Values
If successful, the subroutine returns a handle defined as a pointer to a structure of type cx_handle. If an error occurs, NULL is returned and an error text may be placed in the external character array RSiEMsg.

Error Codes
All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:
- extern char RSiEMsg[];
- extern int RSiErrno;

If the subroutine returns without an error, the RSiErrno variable is set to RSiOkay and the RSiEMsg character array is empty. If an error is detected, the RSiErrno variable returns an error code, as defined in the enum RSiErrorType. RSi error codes are described in List of RSi Error Codes.
RSiStartFeed Subroutine

Purpose
Tells xmservd to start sending data feeds for a statset.

Library
RSI Library (libSpmi.a)

Syntax
#include <sys/Rsi.h>

int RSiStartFeed(rhandle, statset, msecs)

RSiHandle rhandle;
struct SpmiStatSet *statset;
int msecs;

Description
The RSiStartFeed subroutine performs the following function:
1. Informs xmservd of the frequency with which it is required to send data_feed packets.
2. Tells the xmservd to start sending data_feed packets.

This subroutine is part of the Performance Toolbox for AIX licensed product.

Parameters
rhandle Must be an RSiHandle, which was previously initialized by the RSiOpen subroutine.

statset Must be a pointer to a structure of type struct SpmiStatSet, which was previously returned by a successful RSiCreateStatSet subroutine.

msecs The number of milliseconds between the sending of data_feed packets. This number is rounded to a multiple of min_remote_int milliseconds by the xmservd daemon on the remote host. This minimum interval can be modified through the -i command line interval to xmservd.

Return Values
If successful, the subroutine returns zero; otherwise it returns -1 and an error text may be placed in the external character array RSiEMsg.
Error Codes
All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:
- extern char RSiEMsg[];
- extern int RSiErrno;

If the subroutine returns without an error, the RSiErrno variable is set to RSiOkay and the RSiEMsg character array is empty. If an error is detected, the RSiErrno variable returns an error code, as defined in the enum RSiErrorType. RSi error codes are described in [List of RSi Error Codes](#).

Files
/usr/include/sys/Rsi.h
Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

Related Information
For related information, see:
- “RSiCreateStatSet Subroutine” on page 72
- “RSiOpen Subroutine” on page 90
- “RSiStopFeed Subroutine” on page 99.

RSiStartHotFeed Subroutine

Purpose
Tells xmservd to start sending hot feeds for a hotset or to start checking for if exceptions or SNMP traps should be generated.

Library
RSI Library (libSpmi.a)

Syntax
#include sys/Rsi.h
int RSiStartFeed(rhandle, hotset, msecs)
RSiHandle rhandle;
struct SpmiHotSet *hotset;
int msecs;

Description
The RSiStartHotFeed subroutine performs the following function:
1. Informs xmservd of the frequency with which it is required to send hot_feed packets, if the hotset is defined to generate hot_feed packets.
2. Informs xmservd of the frequency with which it is required to check if exceptions or SNMP traps should be generated. This is only done if it is specified for the hotset that exceptions and/or SNMP traps should be generated.
3. Tells the xmservd to start sending data_feed packets and/or start checking for exceptions or traps.

This subroutine is part of the Performance Toolbox for AIX licensed product.
Parameters

- **rhandle**: Must be an `RSiHandle`, which was previously initialized by the [RSiOpen Subroutine](#) subroutine.

- **hotset**: Must be a pointer to a structure of type `struct SpmiHotSet`, which was previously returned by a successful [RSiCreateHot](#) subroutine call.

- **msecs**: The number of milliseconds between the sending of hot_feed packets and/or the number of milliseconds between checks for if exceptions or SNMP traps should be generated. This number is rounded to a multiple of min_remote_int milliseconds by the xmservd daemon on the remote host. This minimum interval can be modified through the -i command line interval to xmservd.

Return Values

If successful, the subroutine returns zero; otherwise it returns -1 and an error text may be placed in the external character array `RSiEMsg`.

Error Codes

All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- `extern char RSiEMsg[];
- `extern int RSiErrno;`

If the subroutine returns without an error, the `RSiErrno` variable is set to `RSiOkay` and the `RSiEMsg` character array is empty. If an error is detected, the `RSiErrno` variable returns an error code, as defined in `RSiErrorType`. RSi error codes are described in [List of RSi Error Codes](#).

Files

- `/usr/include/sys/Rsi.h`: Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

Related Information

For related information, see:

- [RSiCreateHotSet Subroutine](#) on page 71
- [RSiOpen Subroutine](#) on page 90
- [RSiChangeHotFeed Subroutine](#) on page 69
- [RSiStopHotFeed Subroutine](#) on page 100.

---

**RSiStatGetPath Subroutine**

**Purpose**

Finds the full path name of a statistic identified by a `SpmiStatVals` pointer.

**Library**

RSI Library (`libSpmi.a`)

**Syntax**

```c
#include <sys/Rsi.h>
```
char *RSiStatGetPath(rhandle, svp)
RSiHandle rhandle;
struct SpmiStatVals *svp;

Description
The **RSiStatGetPath** subroutine performs the following:
1. Validates that the **SpmiStatVals** statistic identified by the second argument does exist.
2. Returns a pointer to a character array containing the full value path name of the statistic.

The memory area pointed to by the returned pointer is freed when the **RSiStatGetPath** subroutine call is repeated. For each invocation of the subroutine, a new memory area is allocated and its address returned.

If the calling program needs the returned character string after issuing the **RSiStatGetPath** subroutine call, the program must copy the returned string to locally allocated memory before reissuing the subroutine call.

This subroutine is part of the Performance Toolbox for AIX licensed product.

Parameters
**rhandle**Must be an **RSiHandle**, previously initialized by the **RSiOpen** subroutine.

**svp**Must be a handle of type **struct SpmiStatVals** as returned by a successful **RSiPathAddSetStat** subroutine call.

Return Values
If successful, the **RSiStatGetPath** subroutine returns a pointer to a character array containing the full path name of the statistic. If unsuccessful, the subroutine returns a NULL value and an error text may be placed in the external character array **RSiEMsg**.

Error Codes
All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char RSiEMsg[];
- extern int RSiErrno;

If the subroutine returns without an error, the **RSiErrno** variable is set to **RSiOkay** and the **RSiEMsg** character array is empty. If an error is detected, the **RSiErrno** variable returns an error code, as defined in the enum **RSiErrorType**. RSI error codes are described in [List of RSI Error Codes](#).

Files

/usr/include/sys/Rsi.h Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

Related Information
For related information, see:
- [“RSiOpen Subroutine” on page 90](#)
- [“RSiPathAddSetStat Subroutine” on page 93](#)
RSiStopFeed Subroutine

Purpose
Tells xmservd to stop sending data feeds for a statset.

Library
RSI Library (libSpmi.a)

Syntax
```c
#include <sys/Rsi.h>
int RSiStopFeed(rhandle, statset, erase)
RsiHandle rhandle;
struct SpmiStatSet *statset;
boolean erase;
```

Description
The RSiStopFeed subroutine instructs the xmservd of a remote system to:
1. Stop sending data_feed packets for a given SpmiStatSet. If the daemon is not told to erase the SpmiStatSet, feeding of data can be resumed by issuing the RSiStartFeed subroutine call for the SpmiStatSet.
2. Optionally tells the daemon and the API library subroutines to erase all their information about the SpmiStatSet. Subsequent references to the erased SpmiStatSet are not valid.

This subroutine is part of the Performance Toolbox for AIX licensed product.

Parameters
- **rhandle**: Must point to a structure of type RSiHandle, which was previously initialized by the RSiOpen subroutine.
- **statset**: Must be a pointer to a structure of type struct SpmiStatSet, which was previously returned by a successful RSiCreateStatSet subroutine call. Data feeding must have been started for this SpmiStatSet via a previous RSiStartFeed subroutine call.
- **erase**: If this argument is set to true, the xmservd daemon on the remote host discards all information about the named SpmiStatSet. Otherwise the daemon maintains its definition of the set of statistics.

Return Values
If successful, the subroutine returns zero, otherwise -1. A NULL error text is placed in the external character array RSiEMsg regardless of the subroutine’s success or failure.

Error Codes
All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:
- extern char RSiEMsg[];
- extern int RSiErrno;

If the subroutine returns without an error, the RSiErrno variable is set to RSiOkay and the RSiEMsg character array is empty. If an error is detected, the RSiErrno variable returns an error code, as defined in the enum RSiErrorType. RSI error codes are described in List of RSI Error Codes.
RSiStopHotFeed Subroutine

Purpose
Tells xmservd to stop sending hot feeds for a hotset and to stop checking for exception and SNMP trap generation.

Library
RSI Library (libSpmi.a)

Syntax
```c
#include <sys/Rsi.h>
int RSiStopHotFeed(rhandle, hotset, erase)
RSiHandle rhandle;
struct SpmiHotSet *hotset;
boolean erase;
```

Description
The RSiStopHotFeed subroutine instructs the xmservd of a remote system to:
1. Stop sending hot_feed packets or check if exceptions or SNMP traps should be generated for a given SpmiHotSet. If the daemon is not told to erase the SpmiHotSet, feeding of data can be resumed by issuing the RSiStartHotFeed ("RSiStartHotFeed Subroutine" on page 96) subroutine call for the SpmiHotSet.
2. Optionally tells the daemon and the API library subroutines to erase all their information about the SpmiHotSet. Subsequent references to the erased SpmiHotSet are not valid.

This subroutine is part of the Performance Toolbox for AIX licensed product.

Parameters
rhandleMust point to a structure of type RSiHandle, which was previously initialized by the RSiOpen ("RSiOpen Subroutine" on page 90) subroutine.

hotsetMust be a pointer to a structure of type struct SpmiHotSet, which was previously returned by a successful RSiCreateHotSet ("RSiCreateHotSet Subroutine" on page 71) subroutine call. Data feeding must have been started for this SpmiStatSet via a previous RSiStartHotFeed ("RSiStartHotFeed Subroutine" on page 96) subroutine call.

eraseIf this argument is set to true, the xmservd daemon on the remote host discards all information about the named SpmiHotSet. Otherwise the daemon maintains its definition of the set of statistics.
Return Values
If successful, the subroutine returns zero, otherwise -1. A NULL error text is placed in the external character array `RSiEMsg` regardless of the subroutine’s success or failure.

Error Codes
All RSI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:
- extern char `RSiEMsg[]`;
- extern int `RSiErrno`;

If the subroutine returns without an error, the `RSiErrno` variable is set to `RSiOkay` and the `RSiEMsg` character array is empty. If an error is detected, the `RSiErrno` variable returns an error code, as defined in the enum `RSiErrorType`. RSi error codes are described in [List of RSI Error Codes].

Files
`/usr/include/sys/Rsi.h` Declares the subroutines, data structures, handles, and macros that an application program can use to access the RSI.

Related Information
For related information, see:
- “RSiOpen Subroutine” on page 90
- “RSiStartHotFeed Subroutine” on page 96
- “RSiChangeHotFeed Subroutine” on page 69.

rs_alloc Subroutine

Purpose
Allocates a resource set and returns its handle.

Library
Standard C library (`libc.a`)

Syntax
```
#include <sys/rset.h>
rsethandle_t rs_alloc (flags)
unsigned int flags;
```

Description
The `rs_alloc` subroutine allocates a resource set and initializes it according to the information specified by the `flags` parameter. The value of the `flags` parameter determines how the new resource set is initialized.

The handle for the new resource set is returned by the subroutine.
Parameters

flags  Specifies how the new resource set is initialized. It takes one of the following values, defined in rset.h:

- **RS_EMPTY** (or 0 value): The resource set is initialized to contain no resources.
- **RS_SYSTEM**: The resource set is initialized to contain available system resources.
- **RS_ALL**: The resource set is initialized to contain all resources.
- **RS_PARTITION**: The resource set is initialized to contain the resources in the caller’s process partition resource set.

Return Values

On successful completion, a resource set handle for the new resource set is returned. Otherwise, a value of 0 is returned and the *errno* global variable is set to indicate the error.

Error Codes

The *rs_alloc* subroutine is unsuccessful if one or more of the following are true:

- **EINVAL** The *flags* parameter contains an invalid value.
- **ENOMEM** There is not enough space to create the data structures related to the resource set.

Related Information

"rs_free Subroutine" on page 103, "rs_getinfo Subroutine" on page 105, and "rs_init Subroutine" on page 111.

**rs_discardname Subroutine**

Purpose

Discards a resource set definition from the system resource set registry.

Library

Standard C library (*libc.a*)

Syntax

```c
#include <sys/rset.h>

int rs_discardname(char *namespace, char *rsname);
```

Description

The *rs_discardname* subroutine discards from the system global repository the definition of the resource set. The resource set is identified by the *namespace* and *rsname* parameters. The specified resource set is removed from the registry, and can no longer be shared with other applications.

In order to be able to discard a name from the global repository, the calling process must have root authority or attachment privilege, and an effective user ID equal to that of the *rsname* parameter’s creator.

The *rs_discardname* subroutine is used to remove an exclusive *rset*. When an exclusive *rset* is removed, the state of CPUs in that *rset* is modified so that those CPUs can run any work on the system. Root authority is required to remove an exclusive *rset*. See [Exclusive User Processor Resource Sets](#) and the *mrset command* for more information.
Parameters

namespace  Points to a null terminated string corresponding to the name space within which rsname should be found.
rsname  Points to a null terminated string corresponding to the name of a registered resource set to be discarded.

Return Values

If successful, a value of 0 is returned. Otherwise, a value of -1 is returned, and the errno global variable is set to indicate the error.

Error Codes

The rs_discardname subroutine is unsuccessful if one or more of the following are true:

EINVAL  One of the following is true:
  • The rsname parameter contains a null value.
  • The namespace parameter contains a null value.
  • The rsname or namespace parameters point to an invalid name.
  • The name length is null or greater than the RSET_NAME_SIZE constant (defined in rset.h), or the name contains invalid characters.
EPERM  One of the following is true:
  • The calling process has neither root authority nor CAP_NUMA_ATTACH privileges.
  • The calling process has neither the same user ID as the creator of the rsname definition nor root authority.
  • The namespace parameter starts with sys. This name space is reserved for system use.
EFAULT  Invalid address, and/or exceptions outside errno range.

Related Information

"rs_getnameattr Subroutine” on page 106, “rs_registername Subroutine” on page 115, and “rs_getnamedrset Subroutine” on page 108.

The rmrset command

rs_free Subroutine

Purpose
Frees a resource set.

Library
Standard C library (libc.a)

Syntax

#include <sys/rset.h>

void rs_free (rset)
rsethandle_t rset;

Description
The rs_free subroutine frees a resource set identified by the rset parameter. The resource set must have been allocated by the rs_alloc subroutine.
Parameters

**rset**  Specifies the resource set whose memory will be freed.

Related Information

The "rs_alloc Subroutine" on page 101.

rs_getassociativity Subroutine

Purpose

Gets the hardware associativity values for a resource.

Library

Standard C library (libc.a)

Syntax

```c
#include <sys/rset.h>
int rs_getassociativity (type, id, assoc_array, array_size);
unsigned int type;
unsigned int id;
unsigned int *assoc_array;
unsigned int array_size;
```

Description

The **rs_getassociativity** subroutine returns the array of hardware associativity values for a specified resource. This is a special purpose subroutine intended for specialized root applications needing the hardware associativity value information. The **rs_getinfo, rs_getrad, and rs_numrads** subroutines are provided for non-root applications to discover system hardware topology.

The calling process must have root authority to get hardware associativity values.

Parameters

- **type**  Specifies the resource type whose associativity values are requested. The only value supported to retrieve values for a processor is R_PROCS.
- **id**  Specifies the logical resource id whose associativity values are requested.
- **assoc_array**  Specifies the address of an array of unsigned integers to receive the associativity values.
- **array_size**  Specifies the number of unsigned integers in **assoc_array**.

Return Values

If successful, a value of 0 is returned. The **assoc_array** parameter array contains the resource’s associativity values. The first entry in the array indicates the number of associativity values returned. If the hardware system does not provide system topology data, a value of 0 is returned in the first array entry. If unsuccessful, a value of -1 is returned and the **errno** global variable is set to indicate the error.

Error Codes

The **rs_getassociativity** subroutine is unsuccessful if one or more of the following are true:
EINVAL
One of the following occurred:
- The array_size parameter was specified as 0.
- An invalid type parameter was specified.
ENODEV
The resource specified by the id parameter does not exist.
EFAULT
Invalid address.
EPERM
The calling process does not have root authority.

Related Information
"rs_getinfo Subroutine," "rs_getrad Subroutine" on page 110, and "rs_numrads Subroutine" on page 112.

rs_getinfo Subroutine

Purpose
Gets information about a resource set.

Library
Standard C library (libc.a)

Syntax
#include <sys/rset.h>
int rs_getinfo(rset rset, info_type info_type, flags flags);
rsethandle_t rset;
rsinfo_t info_type;
unsigned int flags;

Description
The rs_getinfo subroutine retrieves information about the resource set identified by the rset parameter. Depending on the value of the info_type parameter, the rs_getinfo subroutine returns information about the number of available processors, the number of available memory pools, or the amount of available memory contained in the resource rset. The subroutine can also return global system information such as the maximum system detail level, the symmetric multiprocessor (SMP) and multiple chip module (MCM) system detail levels, and the maximum number of processor or memory pool resources in a resource set.

Parameters
rset Specifies a resource set handle of a resource set the information should be retrieved from. This parameter is not meaningful if the info_type parameter is R_MAXSDL, R_MAXPROCS, R_MAXMEMPS, R_SMPSDL, or R_MCMSDL.
Specifies the type of information being requested. One of the following values (defined in `rset.h`) can be used:

- **R_LGPGDEF**: The number of defined large pages in the resource set is returned in units of megabytes.
- **R_LGPGFREE**: The number of free large pages in the resource set is returned in units of megabytes.
- **R_NUMPROCS**: The number of available processors in the resource set is returned.
- **R_NUMMEMPS**: The number of available memory pools in the resource set is returned.
- **R_MEMSIZE**: The amount of available memory (in MB) contained in the resource set is returned.
- **R_MAXSDL**: The maximum system detail level of the system is returned.
- **R_MAXPROCS**: The maximum number of processors that may be contained in a resource set is returned.
- **R_MAXMEMPS**: The maximum number of memory pools that may be contained in a resource set is returned.
- **R_SMPSDL**: The system detail level that corresponds to the traditional notion of an SMP is returned. A system detail level of 0 is returned if the hardware system does not provide system topology data.
- **R_MCMSDL**: The system detail level that corresponds to resources packaged in an MCM is returned. A system detail level of 0 is returned if the hardware system does not have MCMs or does not provide system topology data.

*flags*  
Reserved for future use. Specify as 0.

### Return Values

If successful, the requested information is returned. If unsuccessful, a value of -1 is returned and the `errno` global variable is set to indicate the error.

### Error Codes

The `rs_getinfo` subroutine is unsuccessful if one or more of the following are true:

- **EINVAL**  
  One of the following is true:
  - The `info_type` parameter specifies an invalid resource type value.
  - The `flags` parameter was not specified as 0.

- **EFAULT**  
  Invalid address.

### Related Information

The "rs_numrads Subroutine" on page 112.

---

**rs_getnameattr Subroutine**

### Purpose

Retrieves the access control information of a resource set definition in the system resource set registry.

### Library

Standard C library (`libc.a`)

### Syntax

```
#include <sys/rset.h>
int rs_getnameattr(char *namespace, char *rsname, rs_attributes_t *attr);
```
Description
The rs_getnameattr subroutine retrieves from the system resource set registry the access control information of the resource set definition specified by the namespace and rsname parameters.

The owner ID, group ID, and access control information of the specified resource set are stored in the structure pointed to by the attr parameter.

Note: No special authority or access permission is required to query this information.

Parameters
namespace Points to a null terminated string corresponding to the name space within which the rsname parameter should be found.
rsname Points to a null terminated string corresponding to the name the information should be retrieved for.
attr Points to an rs_attributes_t structure containing the owner, group, and mode fields, which will be filled by the subroutine. The mode field in the rs_attributes_t structure is used to store the access permissions, and is constructed by logically ORing one or more of the following values, defined in rset.h:
  - RS_IRUSR: Gives read rights to the name’s owner.
  - RS_IWUSR: Gives write rights to the name’s owner.
  - RS_IRGRP: Gives read rights to users of the same group as the name’s owner.
  - RS_IWGRP: Gives write rights to users of the same group as the name’s owner.
  - RS_IROTH: Gives read rights to others.
  - RS_IWOTH: Gives write rights to others.

Read privilege for a user means that the user can retrieve a resource set definition by issuing a call to the rs_getnamedrset subroutine. Write privilege for a user means that the user can redefine a name by issuing another call to the rs_getnamedrset subroutine.

Return Values
If successful, a value of 0 is returned. If unsuccessful, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes
The rs_getnameattr subroutine is unsuccessful if one or more of the following are true:

EINVAL If one of the following is true:
  - The rsname parameter is a null pointer.
  - The namespace parameter is a null pointer.
  - The rsname or namespace parameters point to an invalid name. The name length is 0 or greater than the RSET_NAME_SIZE constant (defined in rset.h), or the rsname parameter contains invalid characters.

ENOENT The rsname parameter could not be found in the name space identified by the namespace parameter.

EFAULT Invalid address.

Related Information
“rs_registername Subroutine” on page 115, “rs_discardname Subroutine” on page 102, and “rs_getnamedrset Subroutine” on page 108.
rs_getnamedrset Subroutine

Purpose
Retrieves the contents of a named resource set from the system resource set registry.

Library
Standard C library (libc.a)

Syntax
#include <sys/rset.h>
int rs_getnamedrset (namespace, rsname, rset);
char *namespace, *rsname;

Description
The rs_getnamedrset subroutine retrieves a resource set definition from the system registry. The namespace and rsname parameters identify the resource set to be retrieved. The rset parameter identifies where the retrieved resource set should be returned. The namespace and rsname parameters identify a previously registered resource set definition.

The calling process must have root authority or read access rights to the resource set definition in order to retrieve it.

The rset parameter must be allocated (using the rs_alloc subroutine) prior to calling the rs_getnamedrset subroutine.

Parameters
namespace Points to a null-terminated string corresponding to the name space within which rsname is found.
rsname Points to a null-terminated string corresponding to the previously registered name of a resource set.
rset Specifies the resource set handle for the resource set that the registered resource set is copied into. The registered resource set is specified by the rsname parameter.

Return Values
If successful, a value of 0 is returned. If unsuccessful, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes
The rs_getnamedrset subroutine is unsuccessful if one or more of the following are true:

EINVAL One of the following is true:
• The rsname parameter is a null pointer.
• The namespace parameter is a null pointer.
• The rsname or namespace parameters point to an invalid name. The name length is 0 or greater than the RSET_NAME_SIZE constant (defined in rset.h), or the rsname parameter contains invalid characters.

ENOENT The rsname parameter could not be found in the name space identified by the namespace parameter.

EPERM The calling process has neither read permission on rsname nor root authority.

EFAULT Invalid address.
Related Information

“rs_alloc Subroutine” on page 101, “rs_registername Subroutine” on page 115, “rs_getnameattr Subroutine” on page 106, and “rs_discardname Subroutine” on page 102.

rs_getpartition Subroutine

Purpose

Gets the partition resource set to which a process is attached.

Library

Standard C library (libc.a)

Syntax

```c
#include <sys/rset.h>
int rs_getpartition (pid, rset);
pid_t pid; rsethandle_t rset;
```

Description

The `rs_getpartition` subroutine returns the partition resource set attached to the specified process. A process ID value of RS_MYSELF indicates the partition resource set attached to the current process is requested.

The return value from the `rs_getpartition` subroutine indicates the type of resource set returned.

A value of RS_PARTITION_RSET indicates the process has a partition resource set that is set explicitly. This may be set with the `rs_setpartition` subroutine or through the use of WLM work classes with resource sets.

A value of RS_DEFAULT_RSET indicates the process did not have an explicitly set partition resource set. The system default resource set is returned.

Parameters

- `pid` Specifies the process ID whose partition `rset` is requested.
- `rset` Specifies the resource set to receive the process’ partition resource set.

Return Values

If successful, a value of RS_PARTITION_RSET, or RS_DEFAULT_RSET is returned. If unsuccessful, a value of -1 is returned and the global `errno` variable is set to indicate the error.

Error Codes

The `rs_getpartition` subroutine is unsuccessful if one or more of the following are true:

- EFAULT Invalid address.
- ESRCH The process identified by the `pid` parameter does not exist.

Related Information

The “ra_getrset Subroutine” on page 21.
rs_getrad Subroutine

Purpose
Returns a system resource allocation domain (RAD) contained in an input resource set.

Library
Standard C library (libc.a)

Syntax
```
#include <sys/rset.h>
int rs_getrad(rset, rad, sdl, index, flags);
```

Description
The `rs_getrad` subroutine returns a system RAD at a specified system detail level and index that is contained in an input resource set. If only some of the resources in the specified system RAD are contained in the input resource set, only the resources in both the system RAD and the input resource set are returned.

The input resource set is specified by the `rset` parameter. The output system RAD is identified by the `rad` parameter.

The system RAD is specified by system detail level `sdl` and index number `index`. If only a portion of the specified RAD is contained in `rset`, only that portion is returned in `rad`.

The `rset` and `rad` parameters must be allocated (using the `rs_alloc` subroutine) prior to calling the `rs_getrad` subroutine.

Parameters
- `rset`: Specifies a resource set handle for the input resource set.
- `rad`: Specifies a resource set handle to receive the desired system RAD (contained in the `rset` parameter).
- `sdl`: Specifies the system detail level of the desired system RAD.
- `index`: Specifies the index of the system RAD that should be returned from among those at the specified `sdl`. This parameter must belong to the \([0, \text{rs_numrads}(rset, sdl, flags) - 1]\) interval.
- `flags`: The following flags (defined in `rset.h`) can be used to modify the default behavior of the `rs_getrad` subroutine. By default, the `rs_getrad` subroutine empties the resource set specified by `rad` before the specified RAD is retrieved.
  - `RS_UNION`: Instead of emptying `rad` before the specified RAD is retrieved, the RAD retrieved is added to the contents of `rad`. On completion, `rad` contains the union of its original contents and the specified RAD.
  - `RS_EXCLUSION`: Instead of emptying `rad` before the specified RAD is retrieved, the resources in the specified RAD that are also in `rad` are removed from `rad`. On return, `rad` contains all the resources it originally contained except those in the specified RAD.

Return Values
If successful, a value of 0 is returned. If unsuccessful, a value of -1 is returned and the `errno` global variable is set to indicate the error.
Error Codes
The rs_getrad subroutine is unsuccessful if one or more of the following are true:

- **EINVAL** One of the following is true:
  - The flags parameter contains an invalid value.
  - The sdl parameter is greater than the maximum system detail level.
  - The RAD specified by the index parameter does not exist at the system detail level specified by the sdl parameter.

- **EFAULT** Invalid address.

Related Information
"rs_numrads Subroutine" on page 112, "rs_getinfo Subroutine" on page 105, and "rs_alloc Subroutine" on page 101.

rs_init Subroutine

Purpose
Initializes a previously allocated resource set.

Library
Standard C library (libc.a)

Syntax
```c
#include <sys/rset.h>
int rs_init (rset, flags)
rsethandle_t rset;
unsigned int flags;
```

Description
The rs_init subroutine initializes a previously allocated resource set. The resource set is initialized according to information specified by the flags parameter.

Parameters
- **rset** Specifies the handle of the resource set to initialize.
- **flags** Specifies how the resource set is initialized. It takes one of the following values, defined in rset.h:
  - **RS_EMPTY**: The resource set is initialized to contain no resources.
  - **RS_SYSTEM**: The resource set is initialized to contain available system resources.
  - **RS_ALL**: The resource set is initialized to contain all resources.
  - **RS_PARTITION**: The resource set is initialized to contain the resources in the caller’s process partition resource set.

Return Values
If successful, a value of 0 is returned. If unsuccessful, a value of -1 is returned, and the errno global variable is set to indicate the error.
Error Codes
The rs_init subroutine is unsuccessful if one or more of the following are true:

EINVAL The flags parameter contains an invalid value.

Related Information
The rs_alloc Subroutine on page 101.

rs_numrads Subroutine

Purpose
Returns the number of system resource allocation domains (RADs) that have available resources.

Library
Standard C library (libc.a)

Syntax

```c
#include <sys/rset.h>
int rs_numrads(rset, sdl, flags);
rsethandle_t rset;
unsigned int sdl;
unsigned int flags;
```

Description
The rs_numrads subroutine returns the number of system RADs at system detail level sdl, that have available resources contained in the resource set identified by the rset parameter.

The number of atomic RADs contained in the rset parameter is returned if the sdl parameter is equal to the maximum system detail level.

Parameters

- **rset**: Specifies the resource set handle for the resource set being queried.
- **sdl**: Specifies the system detail level in which the caller is interested.
- **flags**: Reserved for future use. Specify as 0.

Return Values
If successful, the number of available RADs at system detail level sdl, that have resources contained in the specified resource set is returned. If unsuccessful, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes
The rs_numrads subroutine is unsuccessful if one or more of the following are true:

EINVAL One of the following is true:
- The flags parameter contains an invalid value.
- The sdl parameter is greater than the maximum system detail level.

EFAULT Invalid address.
rs_op Subroutine

Purpose
Performs a set of operations on one or two resource sets.

Library
Standard C library (libc.a)

Syntax
```
#include <sys/rset.h>
int rs_op (command, rset1, rset2, flags, id)
unsigned int command;
rsethandle_t rset1, rset2;
unsigned int flags;
unsigned int id;
```

Description
The `rs_op` subroutine performs the operation specified by the `command` parameter on resource set `rset1` or both resource sets `rset1` and `rset2`. 
Parameters

command Specifies the operation to apply to the resource sets identified by rset1 and rset2. One of the following values, defined in rset.h, can be used:

- **RS_UNION**: The resources contained in either rset1 or rset2 are stored in rset2.
- **RS_INTERSECTION**: The resources that are contained in both rset1 and rset2 are stored in rset2.
- **RS_EXCLUSION**: The resources in rset1 that are also in rset2 are removed from rset2. On completion, rset2 contains all the resources that were contained in rset2 but were not contained in rset1.
- **RS_COPY**: All resources in rset1 whose type is flags are stored in rset2. If rset1 contains no resources of this type, rset2 will be empty. The previous content of rset2 is lost, while the content of rset1 is unchanged.
- **RS_FIRST**: The first resource whose type is flags is retrieved from rset1 and stored in rset2. If rset1 contains no resources of this type, rset2 will be empty.
- **RS_NEXT**: The resource from rset1 whose type is flags and that follows the resource contained in rset2 is retrieved and stored in rset2. If no resource of the appropriate type follows the resource specified in rset2, rset2 will be empty.
- **RS_NEXT_WRAP**: The resource from rset1 whose type is flags and that follows the resource contained in rset2 is retrieved and stored in rset2. If no resource of the appropriate type follows the resource specified in rset2, rset2 will contain the first resource of this type in rset1.
- **RS_ISEMPTY**: Test if resource set rset1 is empty.
- **RS_ISEQUAL**: Test if resource sets rset1 and rset2 are equal.
- **RS_ISCONTAINED**: Test if all resources in resource set rset1 are also contained in resource set rset2.
- **RS_TESTRESOURCE**: Test if the resource whose type is flags and index is id is contained in resource set rset1.
- **RS_ADDRRESOURCE**: Add the resource whose type is flags and index is id to resource set rset1.
- **RS_DELRESOURCE**: Delete the resource whose type is flags and index is id from resource set rset1.
- **RS_STSET**: Constructs an ST resource set by including only one hardware thread per physical processor included in rset1 and stores it in rset2. Only available processors are considered when constructing the ST resource set.

rset1 Specifies the resource set handle for the first of the resource sets involved in the command operation.

rset2 Specifies the resource set handle for the second of the resource sets involved in the command operation. This resource set is also used, on return, to store the result of the operation, and its previous content is lost. The rset2 parameter is ignored on the RS_ISEMPTY, RS_TESTRESOURCE, RS_ADDRRESOURCE, and RS_DELRESOURCE commands.

flags When combined with the RS_COPY command, the flags parameter specifies the type of the resources that will be copied from rset1 to rset2. When combined with an RS_FIRST or an RS_NEXT command, the flags parameter specifies the type of the resource that will be retrieved from rset1. This parameter is constructed by logically ORing one or more of the following values, defined in rset.h:

- **R_PROCS**: processors
- **R_MEMPS**: memory pools
- **R_ALL_RESOURCES**: processors and memory pools

If none of the above are specified for flags, R_ALL_RESOURCES is assumed.

id On the RS_TESTRESOURCE, RS_ADDRRESOURCE, and RS_DELRESOURCE commands, the id parameter specifies the index of the resource to be tested, added, or deleted. This parameter is ignored on the other commands.
Return Values
If successful, the commands RS_ISEMPTY, RS_ISEQUAL, RS_ISCONTAINED, and
RS_TESTRESOURCE return 0 if the tested condition is not met and 1 if the tested condition is met. All
other commands return 0 if successful. If unsuccessful, a value of -1 is returned and the errno global
variable is set to indicate the error.

Error Codes
The rs_op subroutine is unsuccessful if one or more of the following are true:

EINVAL
If one of the following is true:
- rsset1 identifies an invalid resource set.
- rsset2 identifies an invalid resource set.
- command identifies an invalid operation.
- command is RS_NEXT or RS_NEXT_WRAP*, and rsset2 does not contain a single resource.
- command is RS_NEXT or RS_NEXT_WRAP*, and the single resource contained in rsset2 is not
  also contained in rsset1.
- flags identifies an invalid resource type.
- id specifies a resource index that is too large.

EFAULT
Invalid address.

Related Information
The “rs_alloc Subroutine” on page 101.

rs_registername Subroutine

Purpose
Registers a resource set definition in the system resource set registry.

Library
Standard C library (libc.a)

Syntax
#include <sys/rset.h>
int rs_registername(rset, namespace, rsname, mode, command)
  rsethandle_t rset;
  char *namespace, *rsname;
  unsigned int mode, command;

Description
The rs_registername subroutine registers in the system resource registry (within the name space
identified by namespace) the definition of the resource set identified by the rset handle. The
rs_registername subroutine does this by associating with it the name specified by the null terminated
string structure pointed to by rsname.

If rsname does not exist, the owner and group IDs of rsname are set to the caller’s owner and group IDs,
and the access control information for rsname is set according to the mode parameter.

If rsname already exists, its owner and group IDs and its access control information are left unchanged,
and the mode parameter is ignored. This name can be shared with any applications to identify a dedicated
resource set.
Using the *command* parameter, you can ask to overwrite or not to overwrite the *rsname* parameter’s registration if it already exists in the global repository within the name space identified by *namespace*. If *rsname* already exists within the specified name space and the *command* parameter is set to **not overwrite**, an error is reported to the calling process.

The namespace `sysxrset` is reserved for exclusive *rsets*. When an exclusive *rset* is created, the state of CPUs in the *rset* is modified so that those CPUs only run work that is directed to them. See [Exclusive User Processor Resource Sets](#) and the `mkrset` command for more information. Root privilege is required to create or remove an exclusive *rset*. An exclusive *rset* cannot be overwritten.

**Notes:**
1. Registering a resource set definition can only be done by a process that has root authority or CAP_NUMA_ATTACHMENT attachment privilege.
2. Overwriting an existing name’s registration can be done only by a process that has root authority or write access to this name.

An application registered resource set definition is non-persistent. It does not persist over a system boot.

Both the *namespace* and *rsname* parameters may contain up to 255 characters. They must begin with an ASCII alphanumeric character. Only the period (.), minus (-), and underscore (_) characters can be mixed with ASCII alphanumeric characters within these strings. Moreover, the names are case-sensitive, which means there is a difference between uppercase and lowercase letters in resource set names and name spaces.

**Parameters**

- **rset**: Specifies a resource set handle of a resource set a name should be registered for.
- **namespace**: Points to a null terminated string corresponding to the name space within which *rsname* will be registered.
- **rsname**: Points to a null terminated string corresponding to the name registered with the setting of the resource set specified by *rset*.
- **mode**: Specifies the bit pattern that determines the created name access permissions. It is constructed by logically ORing one or more of the following values, defined in `rset.h`:
  - `RS_IRUSR`: Gives read rights to the name’s owner
  - `RS_IWUSR`: Gives write rights to the name’s owner
  - `RS_IRGRP`: Gives read rights to users of the same group as the name’s owner
  - `RS_IWGRP`: Gives write rights to users of the same group as the name’s owner
  - `RS_IROTH`: Gives read rights to others
  - `RS_IWOTH`: Gives write rights to others

Read privilege for a user means that the user can retrieve a resource set definition (by issuing a call to the `rs_getnamedrset` subroutine). Write privilege for a user means that the user can redefine a name (by issuing another call to the `rs_getnamedrset` subroutine).

- **command**: Specifies whether the *rsname* parameter’s registration should be overwritten if it already exists in the global repository. This parameter takes one of the following values, defined in `rset.h`:
  - `RS_REDEFINE`: The *rsname* parameter should be redefined if it already exists in the name space identified by *namespace*. In such a case, the calling process must have write access to *rsname*.
  - `RS_DEFINE`: The *rsname* parameter should not be redefined if it already exists in the name space identified by *namespace*. If this happens, an error is reported to the calling process.

**Return Values**

If successful, a value of 0 is returned. If unsuccessful, a value of -1 is returned and the *errno* global variable is set to indicate the error.
Error Codes

The rs_registername subroutine is unsuccessful if one or more of the following are true:

EINVAL  
If one of the following is true:
- rsname is a null pointer.
- namespace is a null pointer.
- rsname or namespace points to an invalid name. The name length is 0 or greater than the RSET_NAME_SIZE constant (defined in rset.h), or the name contains invalid characters.
- mode identifies an invalid access rights value.
- command identifies an invalid command value.

EEXIST  
The command parameter is set to RS_DEFINE and rsname already exists in the global repository within the name space identified by namespace.

ENOMEM  
There is not enough space to create the data structures related to the registry of this resource set.

EPERM  
If one of the following is true:
- The command parameter is set to RS_REDEFINE and the calling process has neither write access to rsname nor root authority.
- The calling process has neither the attachment privilege nor root authority.
- The namespace parameter starts with sys. This name space is reserved for system use.

EFAULT  
Invalid address, and/or exceptions outside errno range.

Related Information

"rs_getnameattr Subroutine" on page 106, "rs_discardname Subroutine" on page 102, and "rs_getnamedrset Subroutine" on page 108.

rs_setnameattr Subroutine

Purpose
Sets the access control information of a resource set definition in the system resource set registry.

Library
Standard C library (libc.a)

Syntax

```c
#include <sys/rset.h>
int rs_setnameattr (namespace, rsname, command, attr)
char *namespace, *rsname;
unsigned int command;
rs_attributes *attr;
```

Description

The rs_setnameattr subroutine sets (depending on the command value) one or more of the owner, group, or access control information of the system registry resource set definition specified by the namespace and rsname parameters.

The owner ID and/or group ID and/or access control information of the rsname parameter must be supplied in the structure pointed to by the attr parameter.

Notes:
1. In order to be able to set the attributes of a name, the calling process must have root authority or the attachment privilege and an effective user ID equal to that of the rsname parameter's owner.
2. Root authority is required to change the resource set definition owner ID, or to set its group ID outside of the caller’s list of groups.

Parameters

namespace
Points to a null terminated string corresponding to the name space within which rsname should be found.

rsname
Points to a null terminated string corresponding to the name the information should be retrieved for.

command
Specifies which attributes should be changed. This parameter is constructed by logically ORing one or more of the following values, defined in rset.h:
- RS_OWNER: Set owner as specified in the owner field of attr.
- RS_GROUP: Set group as specified in the group field of attr.
- RS_PERM: Set access control information as specified in the mode field of attr.

attr
Points to an rs_attributes_t structure containing the owner, group and mode fields, which will possibly be used by the subroutine for setting attributes. The mode field is used to store the access permissions, and is constructed by logically ORing one or more of the following values, defined in rset.h:
- RS_IRUSR: Gives read rights to the name’s owner
- RS_IWUSR: Gives write rights to the name’s owner
- RS_IRGRP: Gives read rights to users of the same group as the name’s owner
- RS_IWGRP: Gives write rights to users of the same group as the name’s owner
- RS_IROTH: Gives read rights to the others
- RS_IWOTH: Gives write rights to the others

Return Values

If successful, a value of 0 is returned. If unsuccessful, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes

The rs_setnameattr subroutine is unsuccessful if one or more of the following are true:

EINVAL
One of the following is true:
- rsname is a null pointer.
- namespace is a null pointer.
- rsname or namespace point to an invalid name. Name length is 0 or greater than the RSET_NAME_SIZE constant (defined in rset.h), or name contains invalid characters.
- command identifies an invalid command value.
- command includes RS_PERM and the mode field of attr identifies an invalid access rights value.
- attr is a null pointer.

EPERM
One of the following is true:
- The calling process has neither CAP_NUMA_ATTACH attachment privilege nor root authority.
- command includes RS_OWNER and the owner field of attr is different from the caller’s user ID and the caller does not have root authority.
- command includes RS_GROUP, the group field of attr is outside of the caller’s list of groups, and caller does not have root authority.
- The namespace parameter starts with sys. This name space is reserved for system use.

ENOENT
rsname could not be found in the name space identified by namespace.

ENOSPC
Out of file-space blocks.

EFAULT
Invalid address; exceptions outside errno range.

ENOSYS
The rs_setnameattr subroutine is not supported by the system.
Related Information
The “rs_getnameattr Subroutine” on page 106.

rs_setpartition Subroutine

Purpose
Sets the partition resource set of a process.

Library
Standard C library (libc.a)

Syntax
```c
#include <sys/rset.h>
int rs_setpartition(pid_t pid, rsethandle_t rset, flags);
```

Description
The **rs_setpartition** subroutine sets a process’ partition resource set. The subroutine can also be used to remove a process’ partition resource set.

The partition resource set limits the threads in a process to running only on the processors contained in the partition resource set.

The work component is an existing process identified by the process ID. A process ID value of RS_MYSELF indicates the attachment applies to the current process.

The following conditions must be met to set a process’ partition resource set:

- The calling process must have root authority.
- The resource set must contain processors that are available in the system.
- The new partition resource set must be equal to, or a superset of the target process’ effective resource set.
- The target process must not contain any threads that have bindprocessor bindings to a processor.
- The resource set must be a superset of all the threads’ *rset* in the target process.

The **flags** parameter can be set to indicate the policy for using the resources contained in the resource set specified in the **rset** parameter. The only supported policy today is RS_STRSET, which is useful only when the processors of the system are running in simultaneous multi-threading mode. Processors like the POWER5 support simultaneous multi-threading, where each physical processor has two execution engines, called **hardware threads**. Each hardware thread is essentially equivalent to a single CPU, and each is identified as a separate CPU in a resource set. The RS_STRSET flag indicates that the process is to be scheduled with a single-threaded policy; namely, that it should be scheduled on only one hardware thread per physical processor. All the available processors indicated in the resource set are required to specify this flag for exclusive use. A new resource set, called an **ST resource set**, is constructed from the specified resource set and attached to the process according to the following rules:

- All offline processors are ignored.
- If all the hardware threads (CPUs) of a physical processor (when running in simultaneous multi-threading mode, there will be more than one active hardware thread per physical processor) are not included in the specified resource set, the other CPUs of the processor are ignored when constructing the ST resource set.
• Only one CPU (hardware thread) resource per physical processor is included in the ST resource set.

Parameters

**pid**  
Specifies the process ID of the process whose partition resource set is to be set. A value of RS_MYSELF indicates the current process' partition resource set should be set.

**rset**  
Specifies the partition resource set to be set. A value of RS_DEFAULT indicates the process' partition resource set should be removed.

**flags**  
Specifies the policy to use for the rset. A value of RS_STRSET indicates that the process is to be scheduled with a single-threaded policy (only on one hardware thread per physical processor).

Return Values

If successful, a value of 0 is returned. If unsuccessful, a value of -1 is returned, and the **errno** global variable is set to indicate the error.

Error Codes

The **rs_setpartition** subroutine is unsuccessful if one or more of the following are true:

**EINVAL**  
The RS_STRSET **flags** parameter is specified and one or more processors in the **rset** parameter are not assigned for exclusive use.

**ENODEV**  
The resource set specified by the **rset** parameter does not contain any available processors, or the RS_STRSET **flags** parameter is specified and the constructed ST resource set does not have any available processors.

**ESRCH**  
The process identified by the **pid** parameter does not exist.

**EFAULT**  
Invalid address.

**ENOMEM**  
Memory not available.

**EPERM**  
One of the following is true:

- The calling process does not have root authority.
- The process identified by the **pid** parameter has one or more threads with a bindprocessor processor binding.
- The process identified by the **pid** parameter has an effective resource set and the new partition resource set identified by the **rset** parameter does not contain all of the effective resource set's resources.
- One of the threads in the process identified by the **pid** parameter has a thread level resource set, and the new partition resource set identified by the **rset** parameter does not contain all of the thread level resource set's resources.

Related Information

“**rs_getpartition Subroutine**” on page 109 and “**ra_attachrset Subroutine**” on page 10.

rsqrt Subroutine

**Purpose**

Computes the reciprocal of the square root of a number.

**Libraries**

IEEE Math Library (**libm.a**)  

System V Math Library (**libmsaa.a**)
Syntax
#include <math.h>

double rsqrt(double x)

Description
The rsqrt command computes the reciprocal of the square root of a number \( x \); that is, 1.0 divided by the square root of \( x \) \((1.0/\sqrt{x})\). On some platforms, using the rsqrt subroutine is faster than computing 1.0 / sqrt(\( x \)). The rsqrt subroutine uses the same rounding mode used by the calling program.

When using the libm.a library, the rsqrt subroutine responds to special values of \( x \) in the following ways:

- If \( x \) is NaN, then the rsqrt subroutine returns NaN. If \( x \) is a signaling Nan (NaNs), then the rsqrt subroutine returns a quiet NaN and sets the VX and VXSNAN (signaling NaN invalid operation exception) flags in the FPSCR (Floating-Point Status and Control register) to 1.
- If \( x \) is +/- 0.0, then the rsqrt subroutine returns +/- INF and sets the ZX (zero divide exception) flag in the FPSCR to 1.
- If \( x \) is negative, then the rsqrt subroutine returns NaN, sets the errno global variable to EDOM, and sets the VX and VXSQRT (square root of negative number invalid operation exception) flags in the FPSCR to 1.

When using the libmsaa.a library, the rsqrt subroutine responds to special values of \( x \) in the following ways:

- If \( x \) is +/- 0.0, then the rsqrt subroutine returns +/-HUGE_VAL and sets the errno global variable to EDOM. The subroutine invokes the matherr subroutine, which prints a message indicating a singularity error to standard error output.
- If \( x \) is negative, then the rsqrt subroutine returns 0.0 and sets the errno global variable to EDOM. The subroutine invokes the matherr subroutine, which prints a message indicating a domain error to standard error output.

When compiled with libmsaa.a, a program can use the matherr subroutine to change these error-handling procedures.

Parameter
\( x \quad \text{Specifies a double-precision floating-point value.} \)

Return Values
Upon successful completion, the rsqrt subroutine returns the reciprocal of the square root of \( x \).

1.0 \quad \text{If } x \text{ is 1.0.}
+0.0 \quad \text{If } x \text{ is +INF.}

Error Codes
When using either the libm.a or libmsaa.a library, the rsqrt subroutine may return the following error code:

EDOM \quad 
The value of \( x \) is negative.

Related Information
The(matherr) subroutine, sqrt or cbrt \( \text{“sqrt, sqrtf, or sqrtl Subroutine” on page 278} \) subroutine.
rstat Subroutines

Purpose
Gets performance data from remote kernels.

Library
(librpcsvc.a)

Syntax
#include <rpcsvc/rstat.h>
rstat (host, statp)
char *host;
struct statstime *statp;

Description
The rstat subroutine gathers statistics from remote kernels. These statistics are available on items such as paging, swapping and CPU utilization.

Parameters

host Specifies the name of the machine going to be contacted to obtain statistics found in the statp parameter.

statp Contains statistics from host.

Return Values
If successful, the rstat subroutine fills in the statstime for host and returns a value of 0.

Files
/usr/include/rpcsvc/rstat.x

Related Information
The rup command.
The rstatd daemon

scalbn, scalblnf, scalblnl, scalbn, scalbnf, scalbln, or scalb Subroutine

Purpose
Computes the exponent using FLT_RADIX=2.

Syntax
#include <math.h>

double scalbn (x, n)
double x;
long n;

float scalblnf (x, n)
float x;
long n;

122  Technical Reference, Volume 2: Base Operating System and Extensions
Description
The `scalbln`, `scalblnf`, `scalblnl`, `scalbn`, `scalbnf`, and `scalbnl` subroutines compute $x \times \text{FLT\_RADIX}^n$ efficiently, not normally by computing $\text{FLT\_RADIX}^n$ explicitly. For AIX, $\text{FLT\_RADIX} = 2$.

The `scalb` subroutine returns the value of the $x$ parameter times 2 to the power of the $y$ parameter.

An application wishing to check for error situations should set the `errno` global variable to zero and call `feclearexcept(FE_ALL_EXCEPT)` before calling these subroutines. Upon return, if `errno` is nonzero or `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is nonzero, an error has occurred.

Parameters
$x$ Specifies the value to be computed.
$n$ Specifies the value to be computed.

Return Values
Upon successful completion, the `scalbln`, `scalblnf`, `scalblnl`, `scalbn`, `scalbnf`, and `scalbnl` subroutines return $x \times \text{FLT\_RADIX}^n$.

If the result would cause overflow, a range error occurs and the `scalbln`, `scalblnf`, `scalblnl`, `scalbn`, `scalbnf`, and `scalbnl` subroutines return $\pm \text{HUGE\_VAL}$, $\pm \text{HUGE\_VALF}$, and $\pm \text{HUGE\_VALL}$ (according to the sign of $x$) as appropriate for the return type of the function.

If the correct value would cause underflow, and is not representable, a range error may occur, and 0.0 is returned.

If $x$ is NaN, a NaN is returned.

If $x$ is $\pm 0$ or $\pm \text{Inf}$, $x$ is returned.

If $n$ is 0, $x$ is returned.

If the correct value would cause underflow, and is representable, a range error may occur and the correct value is returned.
Error Codes
If the correct value would overflow, the scalb subroutine returns +/-INF (depending on a negative or positive value of the x parameter) and sets errno to ERANGE.

If the correct value would underflow, the scalb subroutine returns a value of 0 and sets errno to ERANGE.

Related Information
“remainder, remainderf, or remainderl Subroutine” on page 52

math.h in AIX 5L Version 5.3 Files Reference.

scandir, scandir64, alphasort or alphasort64 Subroutine

Purpose
Scans or sorts directory contents.

Library
Standard C Library (libc.a)

Syntax
#include <sys/types.h>
#include <sys/dir.h>

int scandir(DirectoryName, NameList, Select, Compare)
char *DirectoryName;
struct dirent * (* NameList [ ]);
int (* Select) (struct dirent *);
int (* Compare) (void *, void *);

int alphasort (Directory1, Directory2)
void *Directory1, *Directory2;

int scandir64(DirectoryName, NameList, Select, Compare)
char * DirectoryName;
struct dirent64 * (* NameList [ ]);
int (* Select) (struct dirent64 *);
int (* Compare)(void *, void *);

int alphasort64 (Directory1, Directory2)
void *Directory1, *Directory2;

Description
The scandir subroutine reads the directory pointed to by the DirectoryName parameter, and then uses the malloc subroutine to create an array of pointers to directory entries. The scandir subroutine returns the number of entries in the array and, through the NameList parameter, a pointer to the array.
The `Select` parameter points to a user-supplied subroutine that is called by the `scandir` subroutine to select which entries to include in the array. The selection routine is passed a pointer to a directory entry and should return a nonzero value for a directory entry that is included in the array. If the `Select` parameter is a null value, all directory entries are included.

The `Compare` parameter points to a user-supplied subroutine. This routine is passed to the `qsort` subroutine to sort the completed array. If the `Compare` parameter is a null value, the array is not sorted. The `alphasort` subroutine provides comparison functions for sorting alphabetically.

The memory allocated to the array can be deallocated by freeing each pointer in the array, and the array itself, with the `free` subroutine.

The `alphasort` subroutine treats `Directory1` and `Directory2` as pointers to `dirent` pointers and alphabetically compares them. This subroutine can be passed as the `Compare` parameter to either the `scandir` subroutine or the `qsort` subroutine, or a user-supplied subroutine can be used.

The `scandir64` subroutine is similar to the `scandir` subroutine except that it returns a pointer to a list of pointers to `struct dirent64` rather than of `struct dirent`.

The `alphasort64` subroutine treats `Directory1` and `Directory2` as pointers to `dirent64` pointers and alphabetically compares them. This subroutine can be passed as the `Compare` parameter to the `scandir64` subroutine, or a user-supplied subroutine can be used.

**Parameters**

- **DirectoryName**
  Points to the directory name.

- **NameList**
  Points to the array of pointers to directory entries.

- **Select**
  Points to a user-supplied subroutine that is called by the `scandir` subroutine to select which entries to include in the array.

- **Compare**
  Points to a user-supplied subroutine that sorts the completed array.

- **Directory1, Directory2**
  Point to `dirent` structures for `alphasort`, or to `dirent64` structures for `alphasort64`.

**Return Values**

The `scandir` subroutine returns the value -1 if the directory cannot be opened for reading or if the `malloc` subroutine cannot allocate enough memory to hold all the data structures. If successful, the `scandir` subroutine returns the number of entries found.

The `alphasort` subroutine returns the following values:

- **Less than 0**
  The `dirent` structure pointed to by the `Directory1` parameter is lexically less than the `dirent` structure pointed to by the `Directory2` parameter.

- **0**
  The `dirent` structures pointed to by the `Directory1` parameter and the `Directory2` parameter are equal.

- **Greater than 0**
  The `dirent` structure pointed to by the `Directory1` parameter is lexically greater than the `dirent` structure pointed to by the `Directory2` parameter.

The `scandir64` and `alphasort64` subroutines return the similar values as `scandir` and `alphasort` subroutines, except that returned pointers associated with a `dirent` structure are now associated with a `dirent64` structure.
Related Information
The malloc, free, realloc, calloc, mallopt, mallinfo, or alloca subroutine, opendir, readdir, telldir, seekdir, rewinddir, closedir, opendir64, readdir64, telldir64, seekdir64, re winddir64, or closedir64 subroutine, qsort (“qsort Subroutine” on page 1) subroutine.

Files, Directories, and File Systems for Programmers in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

**scanf, fscanf, sscanf, or wsscanf Subroutine**

**Purpose**
Converts formatted input.

**Library**
Standard C Library (libc.a)
or (libc128.a)

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

int scanf (const char *Format [, Pointer, ... ])
const char *Format;

int fscanf (FILE *Stream, const char *Format [, Pointer, ... ])
FILE *Stream;
const char *Format;

int sscanf (const char *String, const char *Format [, Pointer, ... ])
const char *String, *Format;

int wsscanf (const wchar_t *wcs, const char *Format [, Pointer, ... ])
const wchar_t *wcs;
const char *Format;
```

**Description**
The scanf, fscanf, sscanf, and wsscanf subroutines read character data, interpret it according to a format, and store the converted results into specified memory locations. If the subroutine receives insufficient arguments for the format, the results are unreliable. If the format is exhausted while arguments remain, the subroutine evaluates the excess arguments but otherwise ignores them.

These subroutines read their input from the following sources:
- **scanf** Reads from standard input (stdin).
- **fscanf** Reads from the Stream parameter.
- **sscanf** Reads from the character string specified by the String parameter.
- **wsscanf** Reads from the wide character string specified by the wcs parameter.

The scanf, fscanf, sscanf, and wsscanf subroutines can detect a language-dependent radix character, defined in the program’s locale (LC_NUMERIC), in the input string. In the C locale, or in a locale that does not define the radix character, the default radix character is a full stop . (period).
Parameters

wcs Specifies the wide-character string to be read.
Stream Specifies the input stream.
String Specifies input to be read.
Pointer Specifies where to store the interpreted data.
Format Contains conversion specifications used to interpret the input. If there are insufficient arguments for the Format parameter, the results are unreliable. If the Format parameter is exhausted while arguments remain, the excess arguments are evaluated as always but are otherwise ignored.

The Format parameter can contain the following:

- Space characters (blank, tab, new-line, vertical-tab, or form-feed characters) that, except in the following two cases, read the input up to the next nonwhite space character. Unless a match in the control string exists, trailing white space (including a new-line character) is not read.
- Any character except a % (percent sign), which must match the next character of the input stream.
- A conversion specification that directs the conversion of the next input field. The conversion specification consists of the following:
  - The % (percent sign) or the character sequence %n$.

Note: The %n$ character sequence is an X/Open numbered argument specifier. Guidelines for use of the %n% specifier are:
  - The value of n in %n$ must be a decimal number without leading 0’s and must be in the range from 1 to the NL_ARGMAX value, inclusive. See the limits.h file for more information about the NL_ARGMAX value. Using leading 0’s (octal numbers) or a larger n value can have unpredictable results.
  - Mixing numbered and unnumbered argument specifications in a format string can have unpredictable results. The only exceptions are %%% (two percent signs) and %* (percent sign, asterisk), which can be mixed with the %n$ form.
  - Referencing numbered arguments in the argument list from the format string more than once can have unpredictable results.

- The optional assignment-suppression character * (asterisk).
- An optional decimal integer that specifies the maximum field width.
- An optional character that sets the size of the receiving variable for some flags. Use the following optional characters:
  - I Long integer rather than an integer when preceding the d, i, or n conversion codes; unsigned long integer rather than unsigned integer when preceding the o, u, or x conversion codes; double rather than float when preceding the e, f, or g conversion codes.
  - Il Long long integer rather than an integer when preceding the d, i, or n conversion codes; unsigned long long integer rather than unsigned integer when preceding the o, u, or x conversion codes.
  - L A long double rather than a float, when preceding the e, f, or g conversion codes; long integer rather than an integer when preceding the d, i, or n conversion codes; unsigned long integer rather than unsigned integer when preceding the o, u, or x conversion codes.
  - h Short integer rather than an integer when preceding the d, i, and n conversion codes; unsigned short integer (half integer) rather than an unsigned integer when preceding the o, u, or x conversion codes.
An optional character that sets the size of the receiving variable for vector data types. Use the following optional characters:

- **v** vector float (four 4-byte float components) when preceding the e, E, f, g, G, a, or A conversion codes; vector signed char (sixteen 1-byte char components) when preceding the c, d, or i conversion codes; vector unsigned char when preceding the o, u, x, or X conversion codes.

- **vl** or **lv** vector signed integer (four 4-byte integer components) when preceding the d or i conversion codes; vector unsigned integer when preceding the o, u, x, or X conversion codes.

- **vh** or **hv** vector signed short (eight 2-byte integer components) when preceding the d or i conversion codes; vector unsigned short when preceding the o, u, x, or X conversion codes.

For any of the preceding specifiers, an optional separator character can be specified immediately preceding the vector size specifier. If no separator is specified, the default separator is a space unless the conversion is c, in which case the default separator is null. The set of supported optional separators are , (comma), ; (semicolon), : (colon), and _ (underscore).

- A conversion code that specifies the type of conversion to be applied.
  The conversion specification takes the form:
  \%[*][width][size]convcode

The results from the conversion are placed in the memory location designated by the Pointer parameter unless you specify assignment suppression with an * (asterisk). Assignment suppression provides a way to describe an input field to be skipped. The input field is a string of nonwhite space characters. It extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion code indicates how to interpret the input field. The corresponding Pointer parameter must be a restricted type. Do not specify the Pointer parameter for a suppressed field. You can use the following conversion codes:

- **%** Accepts a single % (percent sign) input at this point; no assignment or conversion is done. The complete conversion specification should be %%% (three percent signs).

- **d** Accepts an optionally signed decimal integer with the same format as that expected for the subject sequence of the strtol subroutine with a value of 10 for the base parameter. If no size modifier is specified, the Pointer parameter should be a pointer to an integer.

- **i** Accepts an optionally signed integer with the same format as that expected for the subject sequence of the strtol subroutine with a value of 0 for the base parameter. If no size modifier is specified, the Pointer parameter should be a pointer to an integer.

- **u** Accepts an optionally signed decimal integer with the same format as that expected for the subject sequence of the strtoul subroutine with a value of 10 for the base parameter. If no size modifier is specified, the Pointer parameter should be a pointer to an unsigned integer.

- **o** Accepts an optionally signed octal integer with the same format as that expected for the subject sequence of the strtoul subroutine with a value of 8 for the base parameter. If no size modifier is specified, the Pointer parameter should be a pointer to an unsigned integer.

- **x** Accepts an optionally signed hexadecimal integer with the same format as that expected for the subject sequence of the strtoul subroutine with a value of 16 for the base parameter. If no size modifier is specified, the Pointer parameter should be a pointer to an integer.

- **e**, **f**, or **g**
  Accepts an optionally signed floating-point number with the same format as that expected for the subject sequence of the stdtol subroutine. The next field is converted accordingly and stored
through the corresponding parameter; if no size modifier is specified, this parameter should be a pointer to a float. The input format for floating-point numbers is a string of digits, with some optional characteristics:

- It can be a signed value.
- It can be an exponential value, containing a decimal rational number followed by an exponent field, which consists of an E or an e followed by an (optionally signed) integer.
- It can be one of the special values INF, NaNQ, or NaNs. This value is translated into the IEEE-754 value for infinity, quiet NaN, or signaling NaN, respectively.

**p** Matches an unsigned hexadecimal integer, the same as the %p conversion of the printf subroutine. The corresponding parameter is a pointer to a void pointer. If the input item is a value converted earlier during the same program execution, the resulting pointer compares equal to that value; otherwise, the results of the %p conversion are unpredictable.

**n** Consumes no input. The corresponding parameter is a pointer to an integer into which the scanf, fscanf, sscanf, or wscanf subroutine writes the number of characters (including wide characters) read from the input stream. The assignment count returned at the completion of this function is not incremented.

**s** Accepts a sequence of nonwhite space characters (scanf, fscanf, and sscanf subroutines). The wscanf subroutine accepts a sequence of nonwhite-space wide-character codes; this sequence is converted to a sequence of characters in the same manner as the wcstombs subroutine. The Pointer parameter should be a pointer to the initial byte of a char, signed char, or unsigned char array large enough to hold the sequence and a terminating null-character code, which is automatically added.

**S** Accepts a sequence of nonwhite space characters (scanf, fscanf, and sscanf subroutines). This sequence is converted to a sequence of wide-character codes in the same manner as the mbstowcs subroutine. The wscanf subroutine accepts a sequence of nonwhite-space wide character codes. The Pointer parameter should be a pointer to the initial wide character code of an array large enough to accept the sequence and a terminating null wide character code, which is automatically added. If the field width is specified, it denotes the maximum number of characters to accept.

**c** Accepts a sequence of bytes of the number specified by the field width (scanf, fscanf, and sscanf subroutines); if no field width is specified, 1 is the default. The wscanf subroutine accepts a sequence of wide-character codes of the number specified by the field width; if no field width is specified, 1 is the default. The sequence is converted to a sequence of characters in the same manner as the wcstombs subroutine. The Pointer parameter should be a pointer to the initial bytes of an array large enough to hold the sequence; no null byte is added. The normal skip over white space does not occur.

**C** Accepts a sequence of characters of the number specified by the field width (scanf, fscanf, and sscanf subroutines); if no field width is specified, 1 is the default. The wscanf subroutine accepts a sequence of wide-character codes of the number specified by the field width; if no field width is specified, 1 is the default. The sequence is converted to a sequence of wide-character codes in the same manner as the mbstowcs subroutine. The Pointer parameter should be a pointer to the initial wide character code of an array large enough to hold the sequence; no null wide-character code is added.

**[scanset]**

Accepts a nonempty sequence of bytes from a set of expected bytes specified by the scanset variable (scanf, fscanf, and sscanf subroutines). The wscanf subroutine accepts a nonempty sequence of wide-character codes from a set of expected wide-character codes specified by the scanset variable. The sequence is converted to a sequence of characters in the same manner as the wcstombs subroutine. The Pointer parameter should be a pointer to the initial character of a char, signed char, or unsigned char array large enough to hold the sequence and a terminating null byte, which is automatically added. In the scanf, fscanf, and sscanf subroutines, the conversion specification includes all subsequent bytes in the string specified by the Format.
parameter, up to and including the ] (right bracket). The bytes between the brackets comprise the scanset variable, unless the byte after the [ (left bracket) is a ^ (circumflex). In this case, the scanset variable contains all bytes that do not appear in the scanlist between the ^ (circumflex) and the ] (right bracket). In the wsscanf subroutine, the characters between the brackets are first converted to wide character codes in the same manner as the mbtowc subroutine. These wide character codes are then used as described above in place of the bytes in the scanlist. If the conversion specification begins with [ ] or [^], the right bracket is included in the scanlist and the next right bracket is the matching right bracket that ends the conversion specification. You can also:

- Represent a range of characters by the construct First-Last. Thus, you can express [0123456789] as [0-9]. The First parameter must be lexically less than or equal to the Last parameter or else the - (dash) stands for itself. The - also stands for itself whenever it is the first or the last character in the scanset variable.
- Include the ] (right bracket) as an element of the scanset variable if it is the first character of the scanset. In this case it is not interpreted as the bracket that closes the scanset variable. If the scanset variable is an exclusive scanset variable, the ] is preceded by the ^ (circumflex) to make the ] an element of the scanset. The corresponding Pointer parameter should point to a character array large enough to hold the data field and that ends with a null character (\0). The \0 is added automatically.

A scanf conversion ends at the end-of-file (EOF character), the end of the control string, or when an input character conflicts with the control string. If it ends with an input character conflict, the conflicting character is not read from the input stream.

Unless a match in the control string exists, trailing white space (including a new-line character) is not read.

The success of literal matches and suppressed assignments is not directly determinable.

The National Language Support (NLS) extensions to the scanf subroutines can handle a format string that enables the system to process elements of the argument list in variable order. The normal conversion character % is replaced by %n$, where n is a decimal number. Conversions are then applied to the specified argument (that is, the nth argument), rather than to the next unused argument.

The first successful run of the [fgetc, fgets, fread, getc, getchar, gets, scanf, or fscanf subroutine using a stream that returns data not supplied by a prior call to the ungetc ("ungetc or ungetwc Subroutine" on page 470) subroutine marks the st_atime field for update.

Return Values

These subroutines return the number of successfully matched and assigned input items. This number can be 0 if an early conflict existed between an input character and the control string. If the input ends before the first conflict or conversion, only EOF is returned. If a read error occurs, the error indicator for the stream is set, EOF is returned, and the errno global variable is set to indicate the error.

Error Codes

The scanf, fscanf, sscanf, and wsscanf subroutines are unsuccessful if either the file specified by the Stream, String, or wcs parameter is unbuffered or data needs to be read into the file’s buffer and one or more of the following conditions is true:

- EAGAIN  The O_NONBLOCK flag is set for the file descriptor underlying the file specified by the Stream, String, or wcs parameter, and the process would be delayed in the scanf, fscanf, sscanf, or wsscanf operation.
- EBADF   The file descriptor underlying the file specified by the Stream, String, or wcs parameter is not a valid file descriptor open for reading.
- EINTR   The read operation was terminated due to receipt of a signal, and either no data was transferred or a partial transfer was not reported.
Note: Depending upon which library routine the application binds to, this subroutine may return **EINTR**. Refer to the **signal** subroutine regarding **SA_RESTART**.

**EIO**
The process is a member of a background process group attempting to perform a read from its controlling terminal, and either the process is ignoring or blocking the **SIGTTIN** signal or the process group has no parent process.

**EINVAL**
The subroutine received insufficient arguments for the **Format** parameter.

**EILSEQ**
A character sequence that is not valid was detected, or a wide-character code does not correspond to a valid character.

**ENOMEM**
Insufficient storage space is available.

### Related Information

The **atof**, **atoff**, **strtol**, or **strtold** subroutine, **fread** subroutine, **getc**, **fgetc**, **getchar**, or **getw** subroutine, **fgets** subroutine, **getwc**, **fgetwc**, **getwchar**, or **getwc** subroutine, **mbstowcs** subroutine, **mbtowc** subroutine, **printf**, **fprintf**, **sprintf**, **wsprintf**, **vprintf**, **vfprintf**, or **vwsprintf** subroutine, **setlocale** subroutine, **strtol**, **strtoul**, **atol**, or **atoi** subroutine, **ungetc**, **ungetc** subroutine, **wcstombs** subroutine.

In **AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs**.

In **AIX 5L Version 5.3 National Language Support Guide and Reference**.

---

### sched_get_priority_max and sched_get_priority_min Subroutine

#### Purpose
Retrieves priority limits.

#### Library
Standard C Library (**libc.a**)

#### Syntax
```c
#include <sched.h>

int sched_get_priority_max (policy);
int policy;

int sched_get_priority_min (policy);
int policy;
```

#### Description
The **sched_get_priority_max** and **sched_get_priority_min** subroutines return the appropriate maximum or minimum, respectively, for the scheduling policy specified by the **policy** parameter.

The value of the **policy** parameter is one of the scheduling policy values defined in the **sched.h** header file.

#### Parameters

- **policy**
  
  Specifies the scheduling policy.
Return Values
If successful, the `sched_get_priority_max` and `sched_get_priority_min` subroutines return the appropriate maximum or minimum values, respectively. If unsuccessful, they return -1 and set `errno` to indicate the error.

Error Codes
The `sched_get_priority_max` and `sched_get_priority_min` subroutines fail if:

- **EINVAL** The value of the `policy` parameter does not represent a defined scheduling policy.
- **ENOTSUP** This interface does not support processes capable of checkpoint.

Related Information
- “sched_getparam Subroutine,” “sched_getscheduler Subroutine” on page 133, “sched_rr_get_interval Subroutine” on page 134, and “sched_setscheduler Subroutine” on page 136.

**sched_getparam Subroutine**

**Purpose**
Gets scheduling parameters.

**Library**
Standard C Library (`libc.a`)

**Syntax**
```
#include <sched.h>

int sched_getparam(pid_t pid, struct sched_param *param);
```

**Description**
The `sched_getparam` subroutine returns the scheduling parameters of a process specified by the `pid` parameter in the `sched_param` structure.

If a process specified by the `pid` parameter exists, and if the calling process has permission, the scheduling parameters for the process whose process ID is equal to the value of the `pid` parameter are returned.

If the `pid` parameter is zero, the scheduling parameters for the calling process are returned.

**Parameters**
- **pid** Specifies the process for which the scheduling parameters are retrieved.
- **param** Points to the `sched_param` structure.

**Return Values**
Upon successful completion, the `sched_getparam` subroutine returns zero. If the `sched_getparam` subroutine is unsuccessful, -1 is returned and `errno` is set to indicate the error.
Error Codes

The `sched_rr_get_interval` subroutine fails if:

- **EINVAL**: The `param` parameter is null or a bad address.
- **ENOTSUP**: This interface does not support processes capable of checkpoint.
- **EPERM**: The requesting process does not have permission to obtain the scheduling parameters of the specified process.
- **ESRCH**: The `pid` parameter is negative, or no process can be found that corresponds to the one specified by the `pid` parameter.

Related Information

“`sched_getscheduler Subroutine,” “sched_setparam Subroutine” on page 135, and “sched_setscheduler Subroutine” on page 136.

sched_getscheduler Subroutine

**Purpose**

Gets the scheduling policy.

**Library**

Standard C Library (`libc.a`)

**Syntax**

```c
#include <sched.h>

int sched_getscheduler(pid_t pid);
```

**Description**

The `sched_getscheduler` subroutine returns the scheduling policy of the process specified by the `pid` parameter.

The values that can be returned by the `sched_getscheduler` subroutine are defined in the `sched.h` header file.

**Parameters**

- `pid` Specifies the process for which the scheduling policy is retrieved.

**Return Values**

Upon successful completion, the `sched_getscheduler` subroutine returns the scheduling policy of the specified process. If unsuccessful it returns -1 and sets `errno` to indicate the error.

**Error Codes**

The `sched_getscheduler` subroutine fails if:

- **EPERM**: The requesting process does not have permission to determine the scheduling policy of the specified process.
- **ESRCH**: The `pid` parameter is negative, or no process can be found that corresponds to the one specified by the `pid` parameter.
- **ENOTSUP**: This interface does not support processes capable of checkpoint.
Related Information
"sched_getparam Subroutine" on page 132 and "sched_setscheduler Subroutine" on page 136.

sched_rr_get_interval Subroutine

Purpose
Gets the execution time limits.

Library
Standard C Library (libc.a)

Syntax
#include <sched.h>

int sched_rr_get_interval (pid_t pid, struct timespec *interval);

Description
The sched_rr_get_interval subroutine updates the timespec structure referenced by the interval parameter to contain the current execution time limit for the process specified by the pid parameter.

The current execution time limit applies to process made of system-scope pthreads only, and it is the value of the timeslice tunable for the process specified.

If value of the pid parameter is zero, the current execution time limit for the calling process is returned.

Parameters
pid Specifies the process for which the current execution time limit is retrieved.
interval Points to the timespec structure to be updated.

Return Values
If successful, the sched_rr_get_interval subroutine returns zero. Otherwise, it returns -1 and sets errno to indicate the error.

Error Codes
The sched_rr_get_interval subroutine fails if:

EINVAL The param parameter is null or a bad address.
ENOTSUP This interface does not support processes capable of checkpoint.
ESRCH The pid parameter is negative, or no process can be found that corresponds to the one specified by the pid parameter.

Related Information
"sched_getparam Subroutine" on page 132, "sched_get_priority_max and sched_get_priority_min Subroutine" on page 131, "sched_getscheduler Subroutine" on page 133, "sched_setparam Subroutine" on page 135, and "sched_setscheduler Subroutine" on page 136.
sched_setparam Subroutine

Purpose
Sets scheduling parameters.

Library
Standard C Library (libc.a)

Syntax
#include <sched.h>

int sched_setparam(pid, param)
  pid_t pid;
  const struct sched_param *param;

Description
The sched_setparam subroutine sets the scheduling parameters of the process specified by the pid parameter to the values specified by the sched_param structure pointed to by the param parameter. The value of the sched_priority member in the sched_param structure is any integer within the inclusive priority range for the current scheduling policy. Higher numerical values for the priority represent higher priorities.

If a process specified by the pid parameter exists, and if the calling process has permission, the scheduling parameters are set for the process whose process ID is equal to the value of the pid parameter.

If the pid parameter is zero, the scheduling parameters are set for the calling process.

If the caller is favoring a process, it must have SET_PROC_RAC authority. The caller should have the same effective or real user id or BYPASS_DAC_WRITE authority to modify the priority of the process.

Implementations may require the requesting process to have the appropriate authority to set its own scheduling parameters or those of another process.

The target process, whether it is running or not running, is moved to the end of the thread list for its priority.

If the priority of the process specified by the pid parameter is set higher than that of the lowest priority running process and if the specified process is ready to run, the process specified by the pid parameter preempts the lowest priority running process. Similarly, if the process calling the sched_setparam subroutine sets its own priority lower than that of one or more other non-empty process lists, the process that is the head of the highest priority list also preempts the calling process. Thus, the originating process might not receive notification of the completion of the requested priority change until the higher priority process has executed.

Other scheduling policies (such as, SCHED_FIFO2, SCHED_FIFO3, SCHED_FIFO4) behave like fixed priority scheduling policies (such as, SCHED_FIFO and SCHED_RR).

The effect of the sched_setparam subroutine on individual threads is dependent on the scheduling contention scope of the threads:
  • The sched_setparam subroutine has no effect on the scheduling of threads with system scheduling contention scope.
For threads with process scheduling contention scope, the threads' scheduling parameters are not affected. However, the scheduling of these threads with respect to threads in other processes may be dependent on the scheduling parameters of their process, which are governed using the `sched_setparam` subroutine.

If an implementation supports a two-level scheduling model in which library threads are multiplexed on top of several kernel-scheduled entities, the underlying kernel-scheduled entities for the system contention scope threads are not affected by the `sched_setparam` subroutine.

The underlying kernel-scheduled entities for the process contention scope threads will have their scheduling parameters changed to the value specified in the `param` parameter. Kernel-scheduled entities for use by process contention scope threads created after this call completes inherit their scheduling policy and associated scheduling parameters from the process.

The `sched_setparam` subroutine is not atomic with respect to other threads in the process. Threads might continue to execute while this subroutine call is in the process of changing the scheduling policy for the underlying kernel-scheduled entities.

**Parameters**

`pid` Specifies the process for which the scheduling parameter is set.

`param` Points to the `sched_param` structure.

**Return Values**

If successful, the `sched_setparam` subroutine returns zero.

If the `sched_setparam` subroutine is unsuccessful, the priority remains unchanged, and the subroutine returns a value of -1 and sets `errno` to indicate the error.

**Error Codes**

The `sched_setparam` subroutine fails if:

- `EINVAL` One or more of the requested scheduling parameters is outside the range defined for the scheduling policy of the specified process ID.
- `EINVAL` The `param` parameter is null or a bad address.
- `ENOTSUP` This interface does not support processes capable of checkpoint.
- `EPERM` The requesting process does not have permission to set the scheduling parameters for the specified process, or does not have the appropriate authority to invoke the `sched_setparam` subroutine.
- `ESRCH` The `pid` parameter is negative, or no process can be found that corresponds to the one specified by the `pid` parameter.

**Related Information**

- [“sched_getparam Subroutine” on page 132](#), [“sched_getscheduler Subroutine” on page 133](#), and [“sched_setscheduler Subroutine.”](#)

**sched_setscheduler Subroutine**

**Purpose**

Sets the scheduling policy and parameters.
Library
Standard C Library (libc.a)

Syntax
```
#include <sched.h>

int sched_setscheduler(pid_t pid, int policy, const struct sched_param *param);
```

Description
The `sched_setscheduler` subroutine sets the scheduling policy and scheduling parameters of the process specified by the `pid` parameter to the `policy` parameter and the parameters specified in the `sched_param` structure pointed to by `param`, respectively. The value of the `sched_priority` member in the `sched_param` structure is any integer within the inclusive priority range for the scheduling policy.

The possible values for the `policy` parameter are defined in the `sched.h` header file.

If a process specified by the `pid` parameter exists, and if the calling process has permission, the scheduling policy and scheduling parameters are set for the process.

If the `pid` parameter is zero, the scheduling policy and scheduling parameters are set for the calling process.

In order to change a scheduling policy to a fixed priority scheduling policy, the caller must have SET_PROC_RAC authority. When changing the scheduling policy to the SCHED_OTHER scheduling policy, if the former policy was not SCHED_OTHER, the caller must have SET_PROC_RAC authority.

SET_PROC_RAC authority is not needed if the caller wants to defavor a process under the following conditions:
- The `former_policy` process was SCHED_OTHER.
- The new policy is still SCHED_OTHER.
- The new priority is lower than the old priority (the caller wants to defavor the process).
- All the impacted user process-scope threads have a SCHED_OTHER policy.
- The caller should have the same effective or real user id or BYPASS_DAC_WRITE authority.

The `sched_setscheduler` subroutine is successful if it succeeds in setting the scheduling policy and scheduling parameters of the process specified by `pid` to the values specified by the `policy` parameter and the structure pointed to by the `param` parameter, respectively.

The effect of this subroutine on individual threads is dependent on the scheduling contention scope of the following threads:
- The `sched_setscheduler` subroutine has no effect on threads with system scheduling contention scope.
- For threads with process scheduling contention scope, the threads’ scheduling policy and associated parameters are not affected. However, the scheduling of these threads with respect to threads in other processes might be dependent on the scheduling parameters of their process, which are governed using the `sched_setscheduler` subroutine.

If an implementation supports a two-level scheduling model in which library threads are multiplexed on top of several kernel-scheduled entities, the underlying kernel-scheduled entities for the system contention scope threads are not affected by these subroutines.
The underlying kernel-scheduled entities for the process contention scope threads have their scheduling policy and associated scheduling parameters changed to the values specified in the `policy` and `param` parameters, respectively. Kernel-scheduled entities for use by process contention scope threads that are created after this call completes inherit their scheduling policy and associated scheduling parameters from the process.

This subroutine is not atomic with respect to other threads in the process. Threads may continue to execute while this subroutine is in the process of changing the scheduling policy and associated scheduling parameters for the underlying kernel-scheduled entities used by the process contention scope threads.

**Parameters**

- `pid` Specifies the process for which the scheduling policy and parameters are set.
- `policy` Contains the scheduling policy and scheduling parameters settings.
- `param` Points to the `sched_param` structure.

**Return Values**

Upon successful completion, the `sched_setscheduler` subroutine returns the former scheduling policy of the specified process. If the `sched_setscheduler` subroutine fails to complete successfully, the policy and scheduling parameters will remain unchanged, and the subroutine returns -1 and sets `errno` to indicate the error.

**Error Codes**

The `sched_setscheduler` subroutine fails if:

- **EINVAL** The `param` parameter is null or a bad address.
- **ENOTSUP** This interface does not support processes capable of checkpoint.
- **EPERM** The requesting process does not have permission to set either or both of the scheduling parameters or the scheduling policy of the specified process.
- **ESRCH** The `pid` parameter is negative, or no process can be found that corresponds to the one specified by the `pid` parameter.

**Related Information**

“sched_getparam Subroutine” on page 132, “sched_setparam Subroutine” on page 135, and “sched_setscheduler Subroutine” on page 133.

---

**sched_yield Subroutine**

**Purpose**

Yields the processor.

**Library**

Standard C Library (`libc.a`)

**Syntax**

```c
#include <sched.h>

int sched_yield (void);
```
**Description**
The `sched_yield` subroutine forces the running thread to relinquish the processor until it again becomes the head of its thread list. It takes no parameters.

**Return Values**
The `sched_yield` subroutine returns 0 if it completes successfully. Otherwise, it returns -1 and sets `errno` to indicate the error.

**Error Codes**
The `sched_yield` subroutine fails if:

- **ENOTSUP** This interface does not support processes capable of checkpoint.

---

**select Subroutine**

**Purpose**
Checks the I/O status of multiple file descriptors and message queues.

**Library**
Standard C Library (`libc.a`)

**Syntax**
```
#include <sys/time.h>
#include <sys/select.h>
#include <sys/types.h>

int select (Nfdsmsgs, ReadList, WriteList, ExceptList, TimeOut)
int Nfdsmsgs;
struct timeval * TimeOut;
```

**Description**
The `select` subroutine checks the specified file descriptors and message queues to see if they are ready for reading (receiving) or writing (sending), or if they have an exceptional condition pending.

When selecting on an unconnected stream socket, `select` returns when the connection is made. If selecting on a connected stream socket, then the ready message indicates that data can be sent or received. Files descriptors of regular files always select true for read, write, and exception conditions. For more information on sockets, refer to "Understanding Socket Connections" and the related "Checking for Pending Connections Example Program" dealing with pending connections in AIX 5L Version 5.3 Communications Programming Concepts.

The `select` subroutine is also supported for compatibility with previous releases of this operating system and with BSD systems.

On shared memory descriptors, the `select` subroutine returns true.

**Note:** If selecting on a non-blocking socket for both read and write events and if the destination host is unreachable, `select` could show a different behavior due to timing constraints. Refer to the Examples section of this document for further information.
Parameters

Nfdsmsgs
Specifies the number of file descriptors and the number of message queues to check. The low-order 16 bits give the length of a bit mask that specifies which file descriptors to check; the high-order 16 bits give the size of an array that contains message queue identifiers. If either half of the Nfdsmsgs parameter is equal to a value of 0, the corresponding bit mask or array is assumed not to be present.

TimeOut
Specifies either a null pointer or a pointer to a timeval structure that specifies the maximum length of time to wait for at least one of the selection criteria to be met. The timeval structure is defined in the /usr/include/sys/time.h file and it contains the following members:

struct timeval {
  int tv_sec;  /* seconds */
  int tv_usec; /* microseconds */
};

The number of microseconds specified in TimeOut.tv_usec, a value from 0 to 999999, is set to one millisecond if the process does not have root user authority and the value is less than one millisecond.

If the TimeOut parameter is a null pointer, the select subroutine waits indefinitely, until at least one of the selection criteria is met. If the TimeOut parameter points to a timeval structure that contains zeros, the file and message queue status is polled, and the select subroutine returns immediately.

ReadList, WriteList, ExceptList
Specify what to check for reading, writing, and exceptions, respectively. Together, they specify the selection criteria. Each of these parameters points to a sellist structure, which can specify both file descriptors and message queues. Your program must define the sellist structure in the following form:

struct sellist {
  int fdsmask[F]; /* file descriptor bit mask */
  int msgids[M]; /* message queue identifiers */
};

The fdsmask array is treated as a bit string in which each bit corresponds to a file descriptor. File descriptor n is represented by the bit (1 << (n mod bits)) in the array element fdsmask[n / BITS(int)]. (The BITS macro is defined in the values.h file.) Each bit that is set to 1 indicates that the status of the corresponding file descriptor is to be checked.

**Note:** The low-order 16 bits of the Nfdsmsgs parameter specify the number of bits (not elements) in the fdsmask array that make up the file descriptor mask. If only part of the last int is included in the mask, the appropriate number of low-order bits are used, and the remaining high-order bits are ignored. If you set the low-order 16 bits of the Nfdsmsgs parameter to 0, you must not define an fdsmask array in the sellist structure.

Each int of the msgids array specifies a message queue identifier whose status is to be checked. Elements with a value of -1 are ignored. The high-order 16 bits of the Nfdsmsgs parameter specify the number of elements in the msgids array. If you set the high-order 16 bits of the Nfdsmsgs parameter to 0, you must not define a msgids array in the sellist structure.

**Note:** The arrays specified by the ReadList, WriteList, and ExceptList parameters are the same size because each of these parameters points to the same sellist structure type. However, you need not specify the same number of file descriptors or message queues in each. Set the file descriptor bits that are not of interest to 0, and set the extra elements of the msgids array to -1.

You can use the SELLIST macro defined in the sys/select.h file to define the sellist structure. The format of this macro is:

SELLIST(f, m) declarator . . . ;

where f specifies the size of the fdsmask array, m specifies the size of the msgids array, and each declarator is the name of a variable to be declared as having this type.
Return Values

Upon successful completion, the `select` subroutine returns a value that indicates the total number of file descriptors and message queues that satisfy the selection criteria. The `fdsmask` bit masks are modified so that bits set to 1 indicate file descriptors that meet the criteria. The `msgs` arrays are altered so that message queue identifiers that do not meet the criteria are replaced with a value of -1.

The return value is similar to the `Nfdsmsgs` parameter in that the low-order 16 bits give the number of file descriptors, and the high-order 16 bits give the number of message queue identifiers. These values indicate the sum total that meet each of the read, write, and exception criteria. Therefore, the same file descriptor or message queue can be counted up to three times. You can use the `NFDS` and `NMSG` macros found in the `sys/select.h` file to separate out these two values from the return value. For example, if `rc` contains the value returned from the `select` subroutine, `NFDS(rc)` is the number of files selected, and `NMSG(rc)` is the number of message queues selected.

If the time limit specified by the `TimeOut` parameter expires, the `select` subroutine returns a value of 0.

If a connection-based socket is specified in the `Readlist` parameter and the connection disconnects, the `select` subroutine returns successfully, but the `recv` subroutine on the socket will return a value of 0 to indicate the socket connection has been closed.

For nonblocking connection-based sockets, both successful and unsuccessful connections will cause the `select` subroutine to return successfully without any error.

When the connection completes successfully the socket becomes writable, and if the connection encounters an error the socket becomes both readable and writable.

When using the `select` subroutine, you cannot check any pending errors on the socket. You need to call the `getsockopt` subroutine with `SOL_SOCKET` and `SOL_ERROR` to check for a pending error.

If the `select` subroutine is unsuccessful, it returns a value of -1 and sets the global variable `errno` to indicate the error. In this case, the contents of the structures pointed to by the `ReadList`, `WriteList`, and `ExceptList` parameters are unpredictable.

Error Codes

The `select` subroutine is unsuccessful if one of the following are true:

- **EBADF**: An invalid file descriptor or message queue identifier was specified.
- **EAGAIN**: Allocation of internal data structures was unsuccessful.
- **EINVAL**: An invalid value was specified for the `TimeOut` parameter or the `Nfdsmsgs` parameter.
- **EINVAL**: The STREAM or multiplexer referenced by one of the file descriptors is linked (directly or indirectly) downstream from a multiplexer.
- **EFAULT**: The `ReadList`, `WriteList`, `ExceptList`, or `TimeOut` parameter points to a location outside of the address space of the process.

Examples

The following is an example of the behavior of the `select` subroutine called on a non-blocking socket, when trying to connect to a host that is unreachable:

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <fcntl.h>
#include <sys/time.h>
```
#include <errno.h>
#include <stdio.h>

int main()
{
    int sockfd, cnt, i = 1;
    struct sockaddr_in serv_addr;

    bzero((char *)&serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr("172.16.55.25");
    serv_addr.sin_port = htons(102);

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        exit(1);
    if (fcntl(sockfd, F_SETFL, FNONBLOCK) < 0)
        exit(1);
    if (connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0 && errno != EINPROGRESS)
        exit(1);
    for (cnt=0; cnt<2; cnt++) {
        FD_ZERO(&readfds);
        FD_SET(sockfd, &readfds);
        FD_ZERO(&writefds);
        FD_SET(sockfd, &writefds);

        if (select(sockfd + 1, &readfds, &writefds, NULL, NULL) < 0)
            exit(1);
        printf("Iteration %d ===============\n", i);
        printf("FD_ISSET(sockfd, &readfds) == %d\n", FD_ISSET(sockfd, &readfds));
        printf("FD_ISSET(sockfd, &writefds) == %d\n", FD_ISSET(sockfd, &writefds));
        i++;
    }
    return 0;
}

Here is the output of the above program:

Iteration 1 ===============
FD_ISSET(sockfd, &readfds) == 0
FD_ISSET(sockfd, &writefds) == 1

Iteration 2 ===============
FD_ISSET(sockfd, &readfds) == 1
FD_ISSET(sockfd, &writefds) == 1

In the first iteration, select notifies the write event only. In the second iteration, select notifies both the read and write events.

Notes

**FD_SETSIZE** is the #define variable that defines how many file descriptors the various FD macros will use. The default value for **FD_SETSIZE** will vary, depending on the version of AIX. As the number of open files supported has increased, the default value of **FD_SETSIZE** has increased.

In AIX Version 4.3.1, the size increased to 32767 open file descriptors (from 2000 in prior releases). In AIX 5L™ Version 5.2.0, the size increased to 65534 open file descriptors. This value can not be set greater than **OPEN_MAX**, which also varies from one AIX Version to another.

For more information, refer to the /usr/include/sys/time.h file.
The user may override FD_SETSIZE to select a smaller value before including the system header files. This is desirable for performance reasons, because of the overhead in FD_ZERO to zero 65534 bits.

**Performance Issues and Recommended Coding Practices**

The select subroutine can be a very compute intensive system call, depending on the number of open file descriptors used and the lengths of the bit maps used. Do not follow the examples shown in many text books. Most were written when the number of open files supported was small, and thus the bit maps were short. You should avoid the following (where select is being passed FD_SETSIZE as the number of FDs to process):

```c
select(FD_SETSIZE, ....)
```

Performance will be poor if the program uses FD_ZERO and the default FD_SETSIZE. FD_ZERO should not be used in any loops or before each select call. However, using it one time to zero the bit string will not cause problems. If you plan to use this simple programming method, you should override FD_SETSIZE to define a smaller number of FDs. For example, if your process will only open two FDs that you will be selecting on, and there will never be more than a few hundred other FDs open in the process, you should lower FD_SETSIZE to approximately 1024.

Do not pass FD_SETSIZE as the first parameter to select. This specifies the maximum number of file descriptors the system should check for. The program should keep track of the highest FD that has been assigned or use the getdtablesize subroutine to determine this value. This saves passing excessively long bit maps in and out of the kernel and reduces the number of FDs that select must check.

Use the poll system call instead of select. The poll system call has the same functionality as select, but it uses a list of FDs instead of a bit map. Thus, if you are only selecting on a single FD, you would only pass one FD to poll. With select, you have to pass a bit map that is as long as the FD number assigned for that FD. If AIX assigned FD 4000, for example, you would have to pass a bit map 4001 bits long.

**Related Information**

The poll subroutine.

The Input and Output Handling Programmer’s Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

**sem_close Subroutine**

**Purpose**

Closes a named semaphore.

**Library**

Standard C Library (libc.a)

**Syntax**

```c
#include <semaphore.h>

int sem_close (sem_t *sem);
```

**Description**

The sem_close subroutine indicates that the calling process is finished using the named semaphore indicated by the sem parameter. Calling sem_close for an unnamed semaphore (one created by sem_init) returns an error. The sem_close subroutine deallocates (that is, makes available for reuse by a subsequent call to the sem_open subroutine) any system resources allocated by the system.
process attempts subsequent uses of the semaphore pointed to by \texttt{sem}, an error is returned. If the semaphore has not been removed with a successful call to the \texttt{sem\_unlink} subroutine, the \texttt{sem\_close} subroutine has no effect on the state of the semaphore. If the \texttt{sem\_unlink} subroutine has been successfully invoked for the \texttt{name} parameter after the most recent call to \texttt{sem\_open} with the \texttt{O\_CREATE} flag set, when all processes that have opened the semaphore close it, the semaphore is no longer accessible.

\section*{Parameters}
\begin{itemize}
  \item \texttt{sem} \hfill Indicates the semaphore to be closed.
\end{itemize}

\section*{Return Values}
Upon successful completion, 0 is returned. Otherwise, -1 is returned and \texttt{errno} is set to indicate the error.

\section*{Error Codes}
The \texttt{sem\_close} subroutine fails if:
\begin{itemize}
  \item \texttt{EFAULT} \hfill Invalid user address.
  \item \texttt{EINVAL} \hfill The \texttt{sem} parameter is not a valid semaphore descriptor.
  \item \texttt{ENOMEM} \hfill Insufficient memory for the required operation.
  \item \texttt{ENOTSUP} \hfill This function is not supported with processes that have been checkpoint-restart\text{'ed}.
\end{itemize}

\section*{Related Information}
\begin{itemize}
  \item \texttt{"sem\_init Subroutine" on page 146}, \texttt{"sem\_open Subroutine" on page 147}, and \texttt{"sem\_unlink Subroutine" on page 152}.
\end{itemize}

\section*{sem\_destroy Subroutine}

\subsection*{Purpose}
Destroys an unnamed semaphore.

\subsection*{Library}
Standard C Library (\texttt{libc.a})

\subsection*{Syntax}
\begin{verbatim}
#include <semaphore.h>

int sem_destroy (sem_t *sem);
\end{verbatim}

\subsection*{Description}
The \texttt{sem\_destroy} subroutine destroys the unnamed semaphore indicated by the \texttt{sem} parameter. Only a semaphore that was created using the \texttt{sem\_init} subroutine can be destroyed using the \texttt{sem\_destroy} subroutine; calling \texttt{sem\_destroy} with a named semaphore returns an error. Subsequent use of the semaphore \texttt{sem} returns an error until \texttt{sem} is reinitialized by another call to \texttt{sem\_init}. It is safe to destroy an initialized semaphore upon which other threads are currently blocked.

\subsection*{Parameters}
\begin{itemize}
  \item \texttt{sem} \hfill Indicates the semaphore to be closed.
\end{itemize}
Return Values
Upon successful completion, 0 is returned. Otherwise, -1 is returned and *errno* set to indicate the error.

Error Codes
The **sem_destroy** subroutine fails if:

- **EACCES**: Permission is denied to destroy the unnamed semaphore.
- **EFAULT**: Invalid user address.
- **EINVAL**: The *sem* parameter is not a valid semaphore.
- **ENOTSUP**: This function is not supported with processes that have been checkpoint-restart ed.

Related Information
- "sem_init Subroutine" on page 146, and "sem_open Subroutine" on page 147.

**sem_getvalue Subroutine**

**Purpose**

Gets the value of a semaphore.

**Library**

Standard C Library (*libc.a*)

**Syntax**

```c
#include <semaphore.h>

int sem_getvalue (sem_t *restrict sem,
                  int *restrict sval);
```

**Description**

The **sem_getvalue** subroutine updates the location referenced by the *sval* parameter to have the value of the semaphore referenced by the *sem* parameter without affecting the state of the semaphore. The updated value represents an actual semaphore value that occurred at some unspecified time during the call, but it need not be the actual value of the semaphore when it is returned to the calling process.

If the *sem* parameter is locked, the object to which the *sval* parameter points is set to a negative number whose absolute value represents the number of processes waiting for the semaphore at an unspecified time during the call.

**Parameters**

- **sem**: Indicates the semaphore to be retrieved.
- **sval**: Specifies the location where the semaphore value is stored.

**Return Values**

Upon successful completion, the **sem_getvalue** subroutine returns a 0. Otherwise, it returns a -1 and sets *errno* to indicate the error.
Error Codes
The `sem_getvalue` subroutine fails if:

- **EACCES**: Permission is denied to access the unnamed semaphore.
- **EFAULT**: Invalid user address.
- **EINVAL**: The `sem` parameter does not refer to a valid semaphore.
- **ENOMEM**: Insufficient memory for the required operation.
- **ENOTSUP**: This function is not supported with processes that have been checkpoint-restart'ed.

Related Information
- "sem_open Subroutine" on page 147
- "sem_post Subroutine" on page 149
- "sem_trywait and sem_wait Subroutine" on page 151

**sem_init Subroutine**

Purpose
Initializes an unnamed semaphore.

Library
Standard C Library (`libc.a`)

Syntax
```c
#include <semaphore.h>

int sem_init(sem_t *sem, int pshared, unsigned value);
```

Description
The `sem_init` subroutine initializes the unnamed semaphore referred to by the `sem` parameter. The value of the initialized semaphore is contained in the `value` parameter. Following a successful call to the `sem_init` subroutine, the semaphore might be used in subsequent calls to the `sem_wait`, `sem_trywait`, `sem_post`, and `sem_destroy` subroutines. This semaphore remains usable until it is destroyed.

If the `pshared` parameter has a nonzero value, the semaphore is shared between processes. In this case, any process that can access the `sem` parameter can use it for performing `sem_wait`, `sem_trywait`, `sem_post`, and `sem_destroy` operations.

Only the `sem` parameter itself may be used for performing synchronization.

If the `pshared` parameter is zero, the semaphore is shared between threads of the process. Any thread in this process can use the `sem` parameter for performing `sem_wait`, `sem_trywait`, `sem_post`, and `sem_destroy` operations. The use of the semaphore by threads other than those created in the same process returns an error.

Attempting to initialize a semaphore that has been already initialized results in the loss of access to the previous semaphore.

Parameters
- `sem` Specifies the semaphore to be initialized.
Determines whether the semaphore can be shared between processes or not. 

Contains the value of the initialized semaphore.

**Return Values**

Upon successful completion, the `sem_init` subroutine initializes the semaphore in the `sem` parameter. Otherwise, it returns -1 and sets `errno` to indicate the error.

**Error Codes**

The `sem_init` subroutine fails if:

- **EFAULT** Invalid user address.
- **EINVAL** The `value` parameter exceeds `SEM_VALUE_MAX`.
- **ENFILE** Too many semaphores are currently open in the system.
- **ENOMEM** Insufficient memory for the required operation.
- **ENOSPC** A resource required to initialize the semaphore has been exhausted, or the limit on semaphores, `SEM_NSEMS_MAX`, has been reached.
- **ENOTSUP** This function is not supported with processes that have been checkpoint-restart'ed.

**Related Information**

Related to `sem_destroy Subroutine` on page 144, `sem_post Subroutine` on page 149, and `sem_trywait and sem_wait Subroutine` on page 151.

---

**sem_open Subroutine**

**Purpose**

Initializes and opens a named semaphore.

**Library**

Standard C Library (`libc.a`)

**Syntax**

```c
#include <semaphore.h>

int sem_open(const char *name, mode_t, mode_t, value);
```

**Description**

The `sem_open` subroutine establishes a connection between a named semaphore and a process. Following a call to the `sem_open` subroutine with semaphore name `name`, the process may reference the semaphore using the address returned from the call. This semaphore may be used in subsequent calls to the `sem_wait`, `sem_trywait`, `sem_post`, and `sem_close` subroutines. The semaphore remains usable by this process until the semaphore is closed by a successful call to `sem_close`, `_exit`, or one of the `exec` subroutines.

The `name` parameter points to a string naming a semaphore object. The name has no representation in the file system. The `name` parameter conforms to the construction rules for a pathname. It might begin with a slash character, and it must contain at least one character. Processes calling `sem_open()` with the same value of `name` refers to the same semaphore object, as long as that name has not been removed.
If a process makes multiple successful calls to the `sem_open` subroutine with the same value of the `name` parameter, the same semaphore address is returned for each such successful call, provided that there have been no calls to the `sem_unlink` subroutine for this semaphore.

**Parameters**

- **name**: Points to a string naming a semaphore object.
- **oflag**: Controls whether the semaphore is created or merely accessed by the call to the `sem_open` subroutine. The following flag bits may be set in the `oflag` parameter:
  - **O_CREAT**: This flag is used to create a semaphore if it does not already exist. If the `O_CREAT` flag is set and the semaphore already exists, the `O_CREAT` flag has no effect, except as noted under the description of the `O_EXCL` flag. Otherwise, the `sem_open` subroutine creates a named semaphore. The `O_CREAT` flag requires a third and a fourth parameter: `mode`, which is of type `mode_t`, and `value`, which is of type `unsigned`. The semaphore is created with an initial value of `value`. Valid initial values for semaphores are less than or equal to `SEM_VALUE_MAX`.
  - The user ID of the semaphore is set to the effective user ID of the process. The group ID of the semaphore is set to the effective group ID of the process. The permission bits of the semaphore are set to the value of the `mode` parameter except those set in the file mode creation mask of the process. When bits in mode other than file permission bits are set, they have no effect. When bits in mode other than file permission bits are set, they have no effect.
  - After the semaphore named `name` has been created by the `sem_open` subroutine with the `O_CREAT` flag, other processes can connect to the semaphore by calling the `sem_open` subroutine with the same value of `name`.
  - **O_EXCL**: If the `O_EXCL` and `O_CREAT` flags are set, the `sem_open` subroutine fails if the semaphore name exists. The check for the existence of the semaphore and the creation of the semaphore if it does not exist are atomic with respect to other processes executing the `sem_open` subroutine with the `O_EXCL` and `O_CREAT` flags set. If `O_EXCL` is set and `O_CREAT` is not set, `O_EXCL` is ignored. If flags other than `O_CREAT` and `O_EXCL` are specified in the `oflag` parameter, they have no effect.
  - `mode` Specifies the value of the file permission bits. Used with `O_CREAT` to create a message queue.
  - `value` Specifies the initial value. Used with `O_CREAT` to create a message queue.

**Return Values**

Upon successful completion, the `sem_open` subroutine returns the address of the semaphore. Otherwise, it returns a value of `SEM_FAILED` and sets `errno` to indicate the error. The `SEM_FAILED` symbol is defined in the `semaphore.h` header file. No successful return from the `sem_open` subroutine returns the value `SEM_FAILED`.

**Error Codes**

If any of the following conditions occur, the `sem_open` subroutine returns `SEM_FAILED` and sets `errno` to the corresponding value:

- **EACCES**: The named semaphore exists and the permissions specified by `oflag` are denied.
- **EEXIST**: The `O_CREAT` and `O_EXCL` flags are set and the named semaphore already exists.
- **EFAULT**: Invalid user address.
- **EINVAL**: The `sem_open` subroutine is not supported for the given name, or the `O_CREAT` flag was specified in the `oflag` parameter and `value` was greater than `SEM_VALUE_MAX`.
- **EMFILE**: Too many semaphore descriptors are currently in use by this process.
- **ENAMETOOLONG**: The length of the `name` parameter exceeds `PATH_MAX`, or a pathname component is longer than `NAME_MAX`.
**Related Information**

- "semctl Subroutine" on page 153
- "semget Subroutine" on page 156
- "semop Subroutine" on page 158
- "sem_close Subroutine" on page 143
- "sem_getvalue Subroutine" on page 145
- "sem_post Subroutine"
- "sem_trywait and sem_wait Subroutine" on page 151
- "sem_unlink Subroutine" on page 152

---

**sem_post Subroutine**

**Purpose**

Unlocks a semaphore.

**Library**

Standard C Library (libc.a)

**Syntax**

```c
#include <semaphore.h>

int sem_post(sem_t *sem);
```

**Description**

The `sem_post` subroutine unlocks the semaphore referenced by the `sem` parameter by performing a semaphore unlock operation on that semaphore.

If the semaphore value resulting from this operation is positive, no threads were blocked waiting for the semaphore to become unlocked, and the semaphore value is incremented.

If the value of the semaphore resulting from this operation is zero, one of the threads blocked waiting for the semaphore is allowed to return successfully from its call to the `sem_wait` subroutine. If the Process Scheduling option is supported, the thread to be unblocked is chosen in a manner appropriate to the scheduling policies and parameters in effect for the blocked threads. In the case of the schedulers SCHED_FIFO and SCHED_RR, the highest priority waiting thread shall be is unblocked, and if there is more than one highest priority thread blocked waiting for the semaphore, then the highest priority thread that has been waiting the longest is unblocked. If the Process Scheduling option is not defined, the choice of a thread to unblock is unspecified.

If the Process Sporadic Server option is supported, and the scheduling policy is SCHED_SPORADIC, the semantics are the same as SCHED_FIFO in the preceding paragraph.

The `sem_post` subroutine is reentrant with respect to signals and may be invoked from a signal-catching function.

**Parameters**

- `sem` Specifies the semaphore to be unlocked.
Return Values
If successful, the `sem_post` subroutine returns zero. Otherwise, it returns -1 and sets `errno` to indicate the error.

Error Codes
The `sem_post` subroutine fails if:

- **EACCES** Permission is denied to access the unnamed semaphore.
- **EFAULT** Invalid user address.
- **EIDRM** Semaphore was removed during the required operation.
- **EINVAL** The `sem` parameter does not refer to a valid semaphore.
- **ENOMEM** Insufficient memory for the required operation.
- **ENOTSUP** This function is not supported with processes that have been checkpoint-restart’ed.

Related Information
“sem_open Subroutine” on page 147 and “sem_trywait and sem_wait Subroutine” on page 151.

sem_timedwait Subroutine

Purpose
Locks a semaphore (ADVANCED REALTIME).

Syntax
```c
#include <semaphore.h>
#include <time.h>

int sem_timedwait(sem_t *restrict sem,
                  const struct timespec *restrict abs_timeout);
```

Description
The `sem_timedwait()` function locks the semaphore referenced by `sem` as in the `sem_wait()` function. However, if the semaphore cannot be locked without waiting for another process or thread to unlock the semaphore by performing a `sem_post()` function, this wait terminates when the specified timeout expires.

The timeout expires when the absolute time specified by `abs_timeout` passes—as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds `abs_timeout`)—or when the absolute time specified by `abs_timeout` has already been passed at the time of the call.

If the **Timers** option is supported, the timeout is based on the CLOCK_REALTIME clock. If the **Timers** option is not supported, the timeout is based on the system clock as returned by the `time()` function. The resolution of the timeout matches the resolution of the clock on which it is based. The `timespec` data type is defined as a structure in the `<time.h>` header.

The function never fails with a timeout if the semaphore can be locked immediately. The validity of the `abs_timeout` parameter does not need to be checked if the semaphore can be locked immediately.

Application Usage
The `sem_timedwait()` function is part of the **Semaphores** and **Timeouts** options and need not be provided on all implementations.
Return Values
The sem_timedwait() function returns 0 if the calling process successfully performed the semaphore lock operation on the semaphore designated by sem. If the call was unsuccessful, the state of the semaphore remains unchanged, the function returns a value of -1, and errno is set to indicate the error.

Error Codes
The sem_timedwait() function fails if:

- [EFAULT] abs_timeout references invalid memory.
- [EINVAL] The sem argument does not refer to a valid semaphore.
- [EINVAL] The process or thread would have blocked, and the abs_timeout parameter specified a nanoseconds field value less than 0 or greater than or equal to 1000 million.
- [ETIMEDOUT] The semaphore could not be locked before the specified timeout expired.

The sem_timedwait() function might fail if:

- [EDEADLK] A deadlock condition was detected.
- [EINTR] A signal interrupted this function.

Related Information
"sem_post Subroutine" on page 149, "sem_trywait and sem_wait Subroutine," "semctl Subroutine" on page 153, "semget Subroutine" on page 156, "semop Subroutine" on page 158.

sem_trywait and sem_wait Subroutine

Purpose
Locks a semaphore.

Library
Standard C Library (libc.a)

Syntax
#include <semaphore.h>

int sem_trywait (sem_t *sem);
int sem_wait (sem_t *sem);

Description
The sem_trywait subroutine locks the semaphore referenced by the sem parameter only if the semaphore is currently not locked; that is, if the semaphore value is currently positive. Otherwise, it does not lock the semaphore.

The sem_wait subroutine locks the semaphore referenced by the sem parameter by performing a semaphore lock operation on that semaphore. If the semaphore value is currently zero, the calling thread does not return from the call to the sem_wait subroutine until it either locks the semaphore or the call is interrupted by a signal.

Upon successful return, the state of the semaphore will be locked and will remain locked until the sem_post subroutine is executed and returns successfully.
The `sem_wait` subroutine is interruptible by the delivery of a signal.

**Parameters**

`sem` Specifies the semaphore to be locked.

**Return Values**

The `sem_trywait` and `sem_wait` subroutines return zero if the calling process successfully performed the semaphore lock operation. If the call was unsuccessful, the state of the semaphore is unchanged, and the subroutine returns -1 and sets `errno` to indicate the error.

**Error Codes**

The `sem_trywait` and `sem_wait` subroutines fail if:

- **EACCES** Permission is denied to access the unnamed semaphore.
- **EAGAIN** The semaphore was already locked, so it cannot be immediately locked by the `sem_trywait` subroutine.
- **EFAULT** Invalid user address.
- **EIDRM** Semaphore was removed during the required operation.
- **EINVAL** Permission is denied to access the unnamed semaphore.
- **EINVAL** A signal interrupted the subroutine.
- **ENOMEM** Insufficient memory for the required operation.
- **ENOTSUP** This function is not supported with processes that have been checkpoint-restart'ed.

**Related Information**

"sem_open Subroutine" on page 147 and "sem_post Subroutine" on page 149.

---

**sem_unlink Subroutine**

**Purpose**

Removes a named semaphore.

**Library**

Standard C Library (`libc.a`)

**Syntax**

```c
#include <semaphore.h>

int sem_unlink (const char *name);
```

**Description**

The `sem_unlink` subroutine removes the semaphore named by the string `name`.

If the semaphore named by `name` is currently referenced by other processes, then `sem_unlink` has no effect on the state of the semaphore. If one or more processes have the semaphore open when `sem_unlink` is called, destruction of the semaphore is postponed until all references to the semaphore have been destroyed by calls to `sem_close`, `_exit`, or `exec`. Calls to `sem_open` to recreate or reconnect to the semaphore refer to a new semaphore after `sem_unlink` is called.
The **sem_unlink** subroutine does not block until all references have been destroyed, and it returns immediately.

**Parameters**

*name* Specifies the name of the semaphore to be unlinked.

**Return Values**

Upon successful completion, the **sem_unlink** subroutine returns a 0. Otherwise, the semaphore remains unchanged, -1 is returned, and *errno* is set to indicate the error.

**Error Codes**

The **sem_unlink** subroutine fails if:

- **EACCES** Permission is denied to unlink the named semaphore.
- **EFAULT** Invalid user address.
- **ENAMETOOLONG** The length of the *name* parameter exceeds PATH_MAX or a pathname component is longer than NAME_MAX.
- **ENOENT** The named semaphore does not exist.
- **ENOTSUP** This function is not supported with processes that have been checkpoint-restart’ed.

**Related Information**

[“sem_open Subroutine” on page 147](#) and [“sem_close Subroutine” on page 143](#)

---

**semctl Subroutine**

**Purpose**

Controls semaphore operations.

**Library**

Standard C Library (*libc.a*)

**Syntax**

```c
#include <sys/sem.h>

int semctl (SemaphoreID, SemaphoreNumber, Command, arg)
OR
int semctl (SemaphoreID, SemaphoreNumber, Command)

int SemaphoreID;
int SemaphoreNumber;
int Command;
union semun {
    int val;
    struct semid_ds *buf;
    unsigned short *array;
} arg;
```

If the fourth argument is required for the operation requested, it must be of type union semun and explicitly declared as shown above.
Description
The `semctl` subroutine performs a variety of semaphore control operations as specified by the `Command` parameter.

The following limits apply to semaphores:
- Maximum number of semaphore IDs is 4096 for operating system releases before AIX 4.3.2 and 131072 for AIX 4.3.2 and following.
- Maximum number of semaphores per ID is 65,535.
- Maximum number of operations per call by the `semop` subroutine is 1024.
- Maximum number of undo entries per procedure is 1024.
- Maximum semaphore value is 32,767.
- Maximum adjust-on-exit value is 16,384.

Parameters

SemaphoreID
Specifies the semaphore identifier.

SemaphoreNumber
Specifies the semaphore number.

arg.val
Specifies the value for the semaphore for the SETVAL command.

arg.buf
Specifies the buffer for status information for the IPC_STAT and IPC_SET commands.

arg.array
Specifies the values for all the semaphores in a set for the GETALL and SETALL commands.

Command
Specifies semaphore control operations.

The following `Command` parameter values are executed with respect to the semaphore specified by the SemaphoreID and SemaphoreNumber parameters. These operations get and set the values of a `sem` structure, which is defined in the `sys/sem.h` file.

GETVAL
Returns the semval value, if the current process has read permission.

SETVAL
Sets the semval value to the value specified by the arg.val parameter, if the current process has write permission. When this `Command` parameter is successfully executed, the semadj value corresponding to the specified semaphore is cleared in all processes.

GETPID
Returns the value of the sempid field, if the current process has read permission.

GETNCNT
Returns the value of the semncnt field, if the current process has read permission.

GETZCNT
Returns the value of the semzcnt field, if the current process has read permission.

The following `Command` parameter values return and set every semval value in the set of semaphores. These operations get and set the values of a `sem` structure, which is defined in the `sys/sem.h` file.

GETALL
Stores semvals values into the array pointed to by the arg.array parameter, if the current process has read permission.
SETALL
Sets semvals values according to the array pointed to by the arg.array parameter, if the current process has write permission. When this Command parameter is successfully executed, the semadj value corresponding to each specified semaphore is cleared in all processes.

The following Commands parameter values get and set the values of a semid_ds structure, defined in the sys/sem.h file. These operations get and set the values of a sem structure, which is defined in the sys/sem.h file.

IPC_STAT
Obtains status information about the semaphore identified by the SemaphoreID parameter. This information is stored in the area pointed to by the arg.buf parameter.

IPC_SET
Sets the owning user and group IDs, and the access permissions for the set of semaphores associated with the SemaphoreID parameter. The IPC_SET operation uses as input the values found in the arg.buf parameter structure.

IPC_SET sets the following fields:
- sem_perm.uid: User ID of the owner
- sem_perm.gid: Group ID of the owner
- sem_perm.mode: Permission bits only
- sem_perm.cuid: Creator's user ID

IPC_SET can only be executed by a process that has root user authority or an effective user ID equal to the value of the sem_perm.uid or sem_perm.cuid field in the data structure associated with the SemaphoreID parameter.

IPC_RMID
Removes the semaphore identifier specified by the SemaphoreID parameter from the system and destroys the set of semaphores and data structures associated with it. This Command parameter can only be executed by a process that has root user authority or an effective user ID equal to the value of the sem_perm.uid or sem_perm.cuid field in the data structure associated with the SemaphoreID parameter.

Return Values
Upon successful completion, the value returned depends on the Command parameter as follows:

<table>
<thead>
<tr>
<th>Command</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETVAL</td>
<td>Returns the value of the semval field.</td>
</tr>
<tr>
<td>GETPID</td>
<td>Returns the value of the sempid field.</td>
</tr>
<tr>
<td>GETNCNT</td>
<td>Returns the value of the semncnt field.</td>
</tr>
<tr>
<td>GETZCNT</td>
<td>Returns the value of the semzcnt field.</td>
</tr>
<tr>
<td>All Others</td>
<td>Return a value of 0.</td>
</tr>
</tbody>
</table>

If the semctl subroutine is unsuccessful, a value of -1 is returned and the global variable errno is set to indicate the error.

Error Codes
The semctl subroutine is unsuccessful if any of the following is true:

- EINVAL: The SemaphoreID parameter is not a valid semaphore identifier.
- EINVAL: The SemaphoreNumber parameter is less than 0 or greater than or equal to the sem_nsems value.
- EINVAL: The Command parameter is not a valid command.
- EACCES: The calling process is denied permission for the specified operation.
The `Command` parameter is equal to the `SETVAL` or `SETALL` value and the value to which `semval` value is to be set is greater than the system-imposed maximum.

**ERANGE**

The `Command` parameter is equal to the `IPCP_RMD` or `IPC_SET` value and the calling process does not have root user authority or an effective user ID equal to the value of the `sem_perm.uid` or `sem_perm.cuid` field in the data structure associated with the `SemaphoreID` parameter.

**EFAULT**

The `arg.buf` or `arg.array` parameter points outside of the allocated address space of the process.

**EPERM**

The `arg.buf` or `arg.array` parameter points outside of the allocated address space of the process.

**ENOMEM**

The system does not have enough memory to complete the subroutine.

**Related Information**

The `semget` subroutine, `semop` subroutine.

---

**semget Subroutine**

**Purpose**

Gets a set of semaphores.

**Library**

Standard C Library (`libc.a`)

**Syntax**

```c
#include <sys/sem.h>

int semget (Key, NumberOfSemaphores, SemaphoreFlag)

key_t Key;

int NumberOfSemaphores, SemaphoreFlag;
```

**Description**

The `semget` subroutine returns the semaphore identifier associated with the `Key` parameter value.

The `semget` subroutine creates a data structure for the semaphore ID and an array containing the `NumberOfSemaphores` parameter semaphores if one of the following conditions is true:

- The `Key` parameter is equal to the `IPC_PRIVATE` operation.
- The `Key` parameter does not already have a semaphore identifier associated with it, and the `IPC_CREAT` value is set.

Upon creation, the data structure associated with the new semaphore identifier is initialized as follows:

- The `sem_perm.cuid` and `sem_perm.uid` fields are set equal to the effective user ID of the calling process.
- The `sem_perm.cgid` and `sem_perm.gid` fields are set equal to the effective group ID of the calling process.
- The low-order 9 bits of the `sem_perm.mode` field are set equal to the low-order 9 bits of the `SemaphoreFlag` parameter.
- The `sem_nsems` field is set equal to the value of the `NumberOfSemaphores` parameter.
- The `sem_otime` field is set equal to 0 and the `sem_ctime` field is set equal to the current time.

The data structure associated with each semaphore in the set is not initialized. The `semctl` subroutine (with the `Command` parameter values `SETVAL` or `SETALL`) can be used to initialize each semaphore.
If the `Key` parameter value is not **IPC_PRIVATE**, the **IPC_EXCL** value is not set, and a semaphore identifier already exists for the specified `Key` parameter, the value of the `NumberOfSemaphores` parameter specifies the number of semaphores that the current process needs.

If the `NumberOfSemaphores` parameter has a value of 0, any number of semaphores is acceptable. If the `NumberOfSemaphores` parameter is not 0, the `semget` subroutine is unsuccessful if the set contains fewer than the value of the `NumberOfSemaphores` parameter.

The following limits apply to semaphores:
- Maximum number of semaphore IDs is 4096 for operating system releases before AIX 4.3.2, 131072 for releases AIX 4.3.2 through AIX 5.2, and 1048576 for release AIX 5.3 and later.
- Maximum number of semaphores per ID is 65,535.
- Maximum number of operations per call by the `semop` subroutine is 1024.
- Maximum number of undo entries per procedure is 1024.
- Maximum semaphore value is 32,767.
- Maximum adjust-on-exit value is 16,384.

**Parameters**

- **Key**
  - Specifies either the **IPC_PRIVATE** value or an IPC key constructed by the `ftok` subroutine (or a similar algorithm).

- **NumberOfSemaphores**
  - Specifies the number of semaphores in the set.

- **SemaphoreFlag**
  - Constructed by logically ORing one or more of the following values:
    - **IPC_CREAT**
      - Creates the data structure if it does not already exist.
    - **IPC_EXCL**
      - Causes the `semget` subroutine to fail if the **IPC_CREAT** value is also set and the data structure already exists.
    - **S_IRUSR**
      - Permits the process that owns the data structure to read it.
    - **S_IWUSR**
      - Permits the process that owns the data structure to modify it.
    - **S_IRGRP**
      - Permits the group associated with the data structure to read it.
    - **S_IWGRP**
      - Permits the group associated with the data structure to modify it.
    - **S_IROTH**
      - Permits others to read the data structure.
    - **S_IWOTH**
      - Permits others to modify the data structure.

Values that begin with the **S_** prefix are defined in the `sys/mode.h` file and are a subset of the access permissions that apply to files.

**Return Values**

Upon successful completion, the `semget` subroutine returns a semaphore identifier. Otherwise, a value of -1 is returned and the `errno` global variable is set to indicate the error.
Error Codes

The semget subroutine is unsuccessful if one or more of the following conditions is true:

- **EACCES** A semaphore identifier exists for the Key parameter but operation permission, as specified by the low-order 9 bits of the SemaphoreFlag parameter, is not granted.
- **EINVAL** A semaphore identifier does not exist and the NumberOfSemaphores parameter is less than or equal to a value of 0, or greater than the system-imposed value.
- **EINVAL** A semaphore identifier exists for the Key parameter, but the number of semaphores in the set associated with it is less than the value of the NumberOfSemaphores parameter and the NumberOfSemaphores parameter is not equal to 0.
- **ENOENT** A semaphore identifier does not exist for the Key parameter and the IPC_CREAT value is not set.
- **ENOSPC** Creating a semaphore identifier would exceed the maximum number of identifiers allowed systemwide.
- **EEXIST** A semaphore identifier exists for the Key parameter, but both the IPC_CREAT and IPC_EXCL values are set.
- **ENOMEM** There is not enough memory to complete the operation.

Related Information

The ftok subroutine, semctl subroutine, semop subroutine.

The mode.h file.

semop Subroutine

Purpose

Performs semaphore operations.

Library

Standard C Library (libc.a)

Syntax

```c
#include <sys/sem.h>

int semop (SemaphoreID, SemaphoreOperations, NumberOfSemaphoreOperations)
int SemaphoreID;
struct sembuf SemaphoreOperations;
sizet NumberOfSemaphoreOperations;
```

Description

The semop subroutine performs operations on the set of semaphores associated with the semaphore identifier specified by the SemaphoreID parameter. The sembuf structure is defined in the file.

Each sembuf structure specified by the SemaphoreOperations parameter includes the following fields:

- **sem_num** Semaphore number
- **sem_op** Semaphore operation
- **sem_flg** Operation flags

Each semaphore operation specified by the sem_op field is performed on the semaphore specified by the SemaphoreID parameter and the sem_num field. The sem_op field specifies one of three semaphore operations.
1. If the sem_op field is a negative integer and the calling process has permission to alter, one of the following conditions occurs:
   - If the semval variable (see the /usr/include/sys/sem.h file) is greater than or equal to the absolute value of the sem_op field, the absolute value of the sem_op field is subtracted from the semval variable. In addition, if the SEM_UNDO flag is set in the sem_flg field, the absolute value of the sem_op field is added to the semadj value of the calling process for the specified semaphore.
   - If the semval variable is less than the absolute value of the sem_op field and the IPC_NOWAIT value is set in the sem_flg field, the semop subroutine returns immediately.
   - If the semval variable is less than the absolute value of the sem_op field and the IPC_NOWAIT value is not set in the sem_flg field, the semop subroutine increments the semcnt field associated with the specified semaphore and suspends the calling process until one of the following conditions occurs:
     - The value of the semval variable becomes greater than or equal to the absolute value of the sem_op field. The value of the semcnt field associated with the specified semaphore is then decremented, and the absolute value of the sem_op field is subtracted from the semval variable. In addition, if the SEM_UNDO flag is set in the sem_flg field, the absolute value of the sem_op field is added to the semadj value of the calling process for the specified semaphore.
     - The SemaphoreID parameter for which the calling process is awaiting action is removed from the system. When this occurs, the errno global variable is set to the EIDRM flag and a value of -1 is returned.
     - The calling process received a signal that is to be caught. When this occurs, the semop subroutine decrements the value of the semcnt field associated with the specified semaphore. When the semcnt field is decremented, the calling process resumes as prescribed by the sigaction ("sigaction, sigvec, or signal Subroutine" on page 203) subroutine.

2. If the sem_op field is a positive integer and the calling process has alter permission, the value of the sem_op field is added to the semval variable. In addition, if the SEM_UNDO flag is set in the sem_flg field, the value of the sem_op field is subtracted from the calling process's semadj value for the specified semaphore.

3. If the value of the sem_op field is 0 and the calling process has read permission, one of the following occurs:
   - If the semval variable is 0, the semop subroutine returns immediately.
   - If the semval variable is not equal to 0 and IPC_NOWAIT value is set in the sem_flg field, the semop subroutine returns immediately.
   - If the semval variable is not equal to 0 and the IPC_NOWAIT value is set in the sem_flg field, the semop subroutine increments the semcnt field associated with the specified semaphore and suspends execution of the calling process until one of the following occurs:
     - The value of the semval variable becomes 0. When this occurs, the value of the semcnt field associated with the specified semaphore is decremented.
     - The SemaphoreID parameter for which the calling process is awaiting action is removed from the system. If this occurs, the errno global variable is set to the EIDRM error code and a value of -1 is returned.
     - The calling process received a signal that is to be caught. When this occurs, the semop subroutine decrements the value of the semcnt field associated with the specified semaphore. When the semcnt field is decremented, the calling process resumes execution as prescribed by the sigaction subroutine.

The following limits apply to semaphores:
   - Maximum number of semaphore IDs is 4096 for operating system releases before AIX 4.3.2 and 131072 for AIX 4.3.2 and following.
   - Maximum number of semaphores per ID is 65,535.
   - Maximum number of operations per call by the semop subroutine is 1024.
• Maximum number of undo entries per procedure is 1024.
• Maximum capacity of a semaphore value is 32,767 bytes.
• Maximum adjust-on-exit value is 16,384 bytes.

Parameters

SemaphoreID
NumberofSemaphoreOperations
SemaphoreOperations

Specifies the semaphore identifier.
Specifies the number of structures in the array.
Points to an array of structures, each of which specifies a semaphore operation.

Return Values

Upon successful completion, the semop subroutine returns a value of 0. Also, the SemaphoreID parameter value for each semaphore that is operated upon is set to the process ID of the calling process.

If the semop subroutine is unsuccessful, a value of -1 is returned and the errno global variable is set to indicate the error. If the SEM_ORDER flag was set in the sem_flg field for the first semaphore operation in the SemaphoreOperations array, the SEM_ERR value is set in the sem_flg field for the unsuccessful operation.

If the SemaphoreID parameter for which the calling process is awaiting action is removed from the system, the errno global variable is set to the EIDRM error code and a value of -1 is returned.

Error Codes

The semop subroutine is unsuccessful if one or more of the following are true for any of the semaphore operations specified by the SemaphoreOperations parameter. If the operations were performed individually, the discussion of the SEM_ORDER flag provides more information about error situations.

EINVAL The SemaphoreID parameter is not a valid semaphore identifier.
EFBIG The sem_num value is less than 0 or it is greater than or equal to the number of semaphores in the set associated with the SemaphoreID parameter.
E2BIG The NumberofSemaphoreOperations parameter is greater than the system-imposed maximum.
EACCES The calling process is denied permission for the specified operation.
EAGAIN The operation would result in suspension of the calling process, but the IPC_NOWAIT value is set in the sem_flg field.
ENOSPC The limit on the number of individual processes requesting a SEM_UNDO flag would be exceeded.
EINVAL The number of individual semaphores for which the calling process requests a SEM_UNDO flag would exceed the limit.
ERANGE An operation would cause a semval value to overflow the system-imposed limit.
ERANGE An operation would cause a semadj value to overflow the system-imposed limit.
EFAULT The SemaphoreOperations parameter points outside of the address space of the process.
EINVAL A signal interrupted the semop subroutine.
EINVAL The semaphore identifier SemaphoreID parameter has been removed from the system.

Related Information

The exec subroutine, exit subroutine, fork subroutine, semctl (“semctl Subroutine” on page 153) subroutine, semget (“semget Subroutine” on page 156) subroutine, sigaction (“sigaction, sigvec, or signal Subroutine” on page 203) subroutine.
setacldb or endacldb Subroutine

Purpose
Opens and closes the SMIT ACL database.

Library
Security Library (libc.a)

Syntax
```c
#include <usersec.h>
int setacldb(Mode);
int Mode;
int endacldb;
```

Description
These functions may be used to open and close access to the user SMIT ACL database. Programs that call the `getusraclattr` or `getgrpaclattr` subroutines should call the `setacldb` subroutine to open the database and the `endacldb` subroutine to close the database.

The `setacldb` subroutine opens the database in the specified mode, if it is not already open. The open count is increased by 1.

The `endacldb` subroutine decreases the open count by 1 and closes the database when this count goes to 0. Any uncommitted changed data is lost.

Parameters

`Mode` Specifies the mode of the open. This parameter may contain one or more of the following values defined in the `usersec.h` file:

- **S_READ**
  Specifies read access.

- **S_WRITE**
  Specifies update access.

Return Values
The `setacldb` and `endacldb` subroutines return a value of 0 to indicate success. Otherwise, a value of -1 is returned and the `errno` global variable is set to indicate the error.

Error Codes
The `setacldb` subroutine fails if the following is true:

- **EACCES** Access permission is denied for the data request.

Both subroutines return errors from other subroutines.

Security
Security Files Accessed: The calling process must have access to the SMIT ACL data.

Mode File `rw/etc/security/smitacl.user`
Related Information
The `getgrpaclattr`, `nextgrpaci`, or `putgrpaci` subroutine, `getusracmltr`, `nextusraci`, or `putusracmltr` subroutine.

setauthdb or setauthdb_r Subroutine

Purpose
Defines the current administrative domain.

Library
Standard C Library (libc.a)

Syntax
```c
#include <usersec.h>

int setauthdb (New, Old)
  authdb_t *New;
  authdb_t *Old;

int setauthdb_r (New, Old)
  authdb_t *New;
  authdb_t *Old;
```

Description
The `setauthdb` and `setauthdb_r` subroutines set the value of the current administrative domain in the `New` parameter. The `setauthdb` subroutine sets the value of the current process-wide administrative domain. The `setauthdb_r` subroutine sets the administrative domain for the current thread if one has been set. The subroutines return -1 if no administrative domain has been set. The current administrative domain is returned in the `Old` parameter. The `Old` parameter can be a null pointer if the value of the current administrative domain is not wanted.

The administrative domain determines which user and group information databases will be queried by the user and group library functions. The default behavior is to access all of the defined administrative domains. The `setauthdb` subroutine restricts the user and group library functions to the named administrative domains for all threads in the current process. The `setauthdb_r` subroutine restricts the user and group library functions to the named administrative domain for the current thread. The default behavior can be restored by using a null pointer for the value of the `New` parameter or an empty string for the value of the `New` parameter.

The string referenced by the `New` parameter must be the string `files`, `compat` or an administrative domain defined in the `/usr/lib/security/methods.cfg` file. The `New` and `Old` parameters are of type `authdb_t`. The `authdb_t` type is a 16-character array that contains the name of a loadable authentication module.

Note: The `setauthdb` subroutine affects all threads in the current process and can cause unintended results.

Parameters

`New` Pointer to the name of the new database module. The `New` parameter must reference a value module name contained in the `/usr/lib/security/methods.cfg` file, or one of the predefined values (`BUILTIN`, `compat`, or `files`). The empty string can be used to remove the restriction on which modules are used.
Old  

Pointer to where the name of the current module will be stored. A NULL value for the Old parameter can be used if the current name of the database is not wanted.

## Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The module search restriction has been successfully changed.</td>
</tr>
<tr>
<td>-1</td>
<td>The module search restriction could not be changed. The errno variable must be examined to determine the cause of the failure.</td>
</tr>
</tbody>
</table>

## Error Codes

- **EINVAL**  
  The new_auth_db parameter is longer than the permissible length of a stanza in the /usr/lib/security/methods.cfg file (15 characters).  

- **ENOENT**  
  The new_auth_db does not reference a valid stanza in /usr/lib/security/methods.cfg or one of the predefined values.

## Related Information

getauthdb or getauthdb_r Subroutine in AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 1.

### setbuf, setvbuf, setbuffer, or setlinebuf Subroutine

#### Purpose

Assigns buffering to a stream.

#### Library

Standard C Library (libc.a)

#### Syntax

```c
#include <stdio.h>

void setbuf (FILE *Stream, Buffer);
char *Buffer;

int setvbuf (FILE *Stream, char *Buffer, Mode, Size);
int Mode;
size_t Size;

void setbuffer (FILE *Stream, char *Buffer, size_t Size);

void setlinebuf (FILE *Stream);
```

Chapter 1. Base Operating System (BOS) Runtime Services (Q-Z)  163
Description

The **setbuf** subroutine causes the character array pointed to by the `Buffer` parameter to be used instead of an automatically allocated buffer. Use the **setbuf** subroutine after a stream has been opened, but before it is read or written.

If the `Buffer` parameter is a null character pointer, input/output is completely unbuffered.

A constant, `BUFSIZ`, defined in the `stdio.h` file, tells how large an array is needed:

```c
char buf[BUFSIZ];
```

For the **setbuf** subroutine, the `Mode` parameter determines how the `Stream` parameter is buffered:

- `_IOFBF` Causes input/output to be fully buffered.
- `_IOLBF` Causes output to be line-buffered. The buffer is flushed when a new line is written, the buffer is full, or input is requested.
- `_IONBF` Causes input/output to be completely unbuffered.

If the `Buffer` parameter is not a null character pointer, the array it points to is used for buffering. The `Size` parameter specifies the size of the array to be used as a buffer, but all of the `Size` parameter’s bytes are not necessarily used for the buffer area. The constant `BUFSIZ` in the `stdio.h` file is one buffer size. If input/output is unbuffered, the subroutine ignores the `Buffer` and `Size` parameters. The **setbuffer** subroutine, an alternate form of the **setbuf** subroutine, is used after `Stream` has been opened, but before it is read or written. The character array `Buffer`, whose size is determined by the `Size` parameter, is used instead of an automatically allocated buffer. If the `Buffer` parameter is a null character pointer, input/output is completely unbuffered.

The **setbuffer** subroutine is not needed under normal circumstances because the default file I/O buffer size is optimal.

The **setlinebuf** subroutine is used to change the `stdout` or `stderr` file from block buffered or unbuffered to line-buffered. Unlike the **setbuf** and **setbuffer** subroutines, the **setlinebuf** subroutine can be used any time `Stream` is active.

A buffer is normally obtained from the **malloc** subroutine at the time of the first **getc** subroutine or **putc** subroutine on the file, except that the standard error stream, `stderr`, is normally not buffered.

Output streams directed to terminals are always either line-buffered or unbuffered.

**Note:** 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.

The **setbuffer** and **setlinebuf** subroutines are included for compatibility with Berkeley System Distribution (BSD).

**Parameters**

- `Stream` Specifies the input/output stream.
- `Buffer` Points to a character array.
- `Mode` Determines how the `Stream` parameter is buffered.
- `Size` Specifies the size of the buffer to be used.
Return Values
Upon successful completion, `setvbuf` returns a value of 0. Otherwise it returns a nonzero value if a value that is not valid is given for type, or if the request cannot be honored.

Related Information
The `open`, `fopen`, or `fdopen` subroutine, `read` subroutine, `getc`, `fgetc`, `getchar`, or `getw` subroutine, `fgetwc`, `fgetwc`, or `getwc` subroutine, `malloc`, `free`, `realloc`, `calloc`, `malloc`, `mallinfo`, or `alloca` subroutine, `putc`, `putchar`, `fputc`, or `putw` subroutine, `putwc`, `putwchar`, or `fputwc` subroutine.

The Input and Output Handling in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

setcsmap Subroutine

Purpose
Reads a code-set map file and assigns it to the standard input device.

Library
Standard C Library (`libc.a`)

Syntax
```
#include <sys/termios.h>

int setcsmap (Path);
char * Path;
```

Description
The `setcsmap` subroutine reads in a code-set map file. The `Path` parameter specifies the location of the code-set map file. The path is usually composed by forming a string with the `csmap` directory and the code set, as in the following example:
```
n=sprintf(path,"%s%"CSMAP_DIR,nl_langinfo(CODESET));
```

The file is processed and according to the included informations, the `setcsmap` subroutine changes the tty configuration. Multibyte processing may be enabled, and converter modules may be pushed onto the tty stream.

Parameter
`Path` Names the code-set map file.

Return Values
If a code set-map file is successfully opened and compiled, a value of 0 is returned. If an error occurred, a value of 1 is returned and the `errno` global variable is set to identify the error.

Error Codes
- **EINVAL** Indicates an invalid value in the code set map.
- **EIO** An I/O error occurred while the file system was being read.
- **ENOMEM** Insufficient resources are available to satisfy the request.
- **EFAULT** A kernel service, such as `copyin`, has failed.
ENOENT  The named file does not exist.
EACCESS The named file cannot be read.

Related Information
The \texttt{setmaps} command.
The \texttt{setmaps} file format.

TTY Subsystem Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

\textbf{setea Subroutine}

\textbf{Purpose}
Sets an extended attribute value.

\textbf{Syntax}
\verb+#include <sys/ea.h>#

\begin{verbatim}
int setea(const char *path, const char *name, void *value, size_t size, int flags);
int fsetea(int filedes, const char *name, void *value, size_t size, int flags);
int lsetea(const char *path, const char *name, void *value, size_t size, int flags);
\end{verbatim}

\textbf{Description}
Extended attributes are name:value pairs associated with the file system objects (such as files, directories, and symlinks). They are extensions to the normal attributes that are associated with all objects in the file system (that is, the \texttt{stat(2)} data).

The \texttt{setea} subroutine sets the value of the extended attribute identified by \texttt{name} and associated with the given \texttt{path} in the file system. The size of the value must be specified. The \texttt{fsetea} subroutine is identical to \texttt{setea}, except that it takes a file descriptor instead of a path. The \texttt{lsetea} subroutine is identical to \texttt{setea}, except, in the case of a symbolic link, the link itself is interrogated rather than the file that it refers to.

\textbf{Parameters}
\begin{itemize}
    \item \texttt{path} \hspace{1cm} The path name of the file.
    \item \texttt{name} \hspace{1cm} The name of the extended attribute. An extended attribute name is a NULL-terminated string.
    \item \texttt{value} \hspace{1cm} A pointer to the value of an attribute. The value of an extended attribute is an opaque byte stream of specified length.
    \item \texttt{size} \hspace{1cm} The length of the value.
    \item \texttt{filedes} \hspace{1cm} A file descriptor for the file.
    \item \texttt{flags} \hspace{1cm} None are defined at this time.
\end{itemize}

\textbf{Return Values}
If the \texttt{setea} subroutine succeeds, 0 is returned. Upon failure, -1 is returned and \texttt{errno} is set appropriately.
Error Codes

EACCES  Caller lacks write permission to the base file, or lacks the appropriate ACL privileges for named attribute write.

EDQUOT  Because of quota enforcement, the remaining space is insufficient to store the extended attribute.

EFAULT  A bad address was passed for path, name, or value.

EFORMAT  File system is capable of supporting EAs, but EAs are disabled.

EINVAL  No flags should be specified.

ENAMETOOLONG  A path-like name should not be used (such as zml/file, ., and ..).

ENOSPC  The remaining space is insufficient to store the extended attribute.

ENOTSUP  Extended attributes are not supported by the file system.

The errors documented for the stat(2) system call are also applicable here.

Related Information

getaea Subroutine  listea Subroutine  "removeea Subroutine" on page 53  "statea Subroutine" on page 314

setgid, setregid, setegid, setregid, or setgidx Subroutine

Purpose
Sets the process group IDs.

Library
Standard C Library (libc.a)

Syntax

#include <unistd.h>

int setgid (GID)  
gid_t GID;

int setregid (RGID)  
gid_t RGID;

int setegid (EGID)  
gid_t EGID;

#include <unistd.h>
#include <sys/id.h>

int setgidx (which, GID)  
int which;

gid_t GID;

Description
The setgid, setregid, setegid, setregid, and setgidx subroutines set the process group IDs of the calling process. The following semantics are supported:
setgid  If the effective user ID of the process is the root user, the process’s real, effective, and saved group IDs are set to the value of the GID parameter. Otherwise, the process effective group ID is reset if the GID parameter is equal to either the current real or saved group IDs, or one of its supplementary group IDs. Supplementary group IDs of the calling process are not changed.

setegid  The process effective group ID is reset if one of the following conditions is met:
- The EGID parameter is equal to either the current real or saved group IDs.
- The EGID parameter is equal to one of its supplementary group IDs.
- The effective user ID of the process is the root user.

setrgid  The EPERM error code is always returned.

setregid  The RGID and EGID parameters can have one of the following relationships:
- RGID != EGID
  If the EGID parameter is equal to either the process’s real or saved group IDs, the process effective group ID is set to the EGID parameter. Otherwise, the EPERM error code is returned.
- RGID == EGID
  If the effective user ID of the process is the root user, the process’s real and effective group IDs are set to the EGID parameter. If the EGID parameter is equal to the process’s real or saved group IDs, the process effective group ID is set to EGID. Otherwise, the EPERM error code is returned.

setgidx  The which parameter can have one of the following values:
- ID_EFFECTIVE
  GID must be either the real or saved GID or one of the values in the concurrent group set. The effective group ID for the current process will be set to GID.
- ID_EFFECTIVE|ID_REAL
  Invoker must have appropriate privilege. The real and effective group ID for the current process will be set to GID.
- ID_EFFECTIVE|ID_REAL|ID_SAVED
  Invoker must have appropriate privilege. The real, effective and saved group ID for the current process will be set to GID.

The setgid, setrgid, setregid, and setgidx subroutines are thread-safe.

The operating system does not support setuid ("setuid, setruid, seteuid, setreuid or setuidx Subroutine" on page 187) or setgid shell scripts.

These subroutines are part of Base Operating System (BOS) Runtime.

Parameters

- GID  Specifies the value of the group ID to set.
- RGID  Specifies the value of the real group ID to set.
- EGID  Specifies the value of the effective group ID to set.
- which  Specifies which group ID values to set.

Return Values

- 0  Indicates that the subroutine was successful.
- -1  Indicates the subroutine failed. The errno global variable is set to indicate the error.
Error Codes
If the setgid, setegid, or setgidx subroutine fails, one or more of the following are returned:

- **EPERM** Indicates the process does not have appropriate privileges and the GID or EGID parameter is not equal to either the real or saved group IDs of the process.
- **EINVAL** Indicates the value of the GID, EGID or which parameter is invalid.

Related Information
The setgid subroutine, getgroups subroutine, setgroups subroutine, setuid subroutine. "setuid, setreuid, setruid, seteuid or setreuidx Subroutine" on page 187 subroutine.

The setgroups command.

List of Security and Auditing Subroutines, Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

setgroups Subroutine

Purpose
Sets the supplementary group ID of the current process.

Library
Standard C Library (libc.a)

Syntax
```c
#include <grp.h>

int setgroups ( NumberGroups, GroupIDSet);
int NumberGroups;
gid_t *GroupIDSet;
```

Description
The setgroups subroutine sets the supplementary group ID of the process. The setgroups subroutine cannot set more than NGROUPS_MAX groups in the group set. (NGROUPS_MAX is a constant defined in the limits.h file.)

Note: The routine may coredump instead of returning EFAULT when an invalid pointer is passed in case of 64-bit application calling 32-bit kernel interface.

Parameters
- **GroupIDSet** Pointer to the array of group IDs to be established.
- **NumberGroups** Indicates the number of entries in the GroupIDSet parameter.

Return Values
Upon successful completion, the setgroups subroutine returns a value of 0. Otherwise, a value of -1 is returned and the errno global variable is set to indicate the error.
**Error Codes**

The **setgroups** subroutine fails if any of the following are true:

- **EFAULT** The `NumberGroups` and `GroupIDSet` parameters specify an array that is partially or completely outside of the process' allocated address space.
- **EINVAL** The `NumberGroups` parameter is greater than the `NGROUPS_MAX` value.
- **EPERM** A group ID in the `GroupIDSet` parameter is not presently in the supplementary group ID, and the invoker does not have root user authority.

**Security**

Auditing Events:

<table>
<thead>
<tr>
<th>Event</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC_SetGroups</td>
<td><code>NumberGroups</code>, <code>GroupIDSet</code></td>
</tr>
</tbody>
</table>

**Related Information**

The `getgid` subroutine, `getgroups` subroutine, `initgroups` subroutine, `setgid` (**setgid, setregid, setegid, setgids, or setgidsx Subroutine** on page 167) subroutine.

List of Security and Auditing Subroutines and Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

**setjmp or longjmp Subroutine**

**Purpose**

Saves and restores the current execution context.

**Library**

Standard C Library (**libc.a**)

**Syntax**

```c
#include <setjmp.h>

int setjmp (Context)
    jmp_buf Context;

void longjmp (Context, Value)
    jmp_buf Context;
    int Value;

int _setjmp (Context)
    jmp_buf Context;

void _longjmp (Context, Value)
    jmp_buf Context;
    int Value;
```

**Description**

The **setjmp** subroutine and the **longjmp** subroutine are useful when handling errors and interrupts encountered in low-level subroutines of a program.

The **setjmp** subroutine saves the current stack context and signal mask in the buffer specified by the `Context` parameter.
The **longjmp** subroutine restores the stack context and signal mask that were saved by the **setjmp** subroutine in the corresponding **Context** buffer. After the **longjmp** subroutine runs, program execution continues as if the corresponding call to the **setjmp** subroutine had just returned the value of the **Value** parameter. The subroutine that called the **setjmp** subroutine must not have returned before the completion of the **longjmp** subroutine. The **setjmp** and **longjmp** subroutines save and restore the signal mask **sigmask** (2), while **_setjmp** and **_longjmp** manipulate only the stack context.

If a process is using the AT&T System V **sigset** interface, then the **setjmp** and **longjmp** subroutines do not save and restore the signal mask. In such a case, their actions are identical to those of the **_setjmp** and **_longjmp** subroutines.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Context</strong></td>
<td>Specifies an address for a <strong>jmp_buf</strong> structure.</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>Indicates any integer value.</td>
</tr>
</tbody>
</table>

**Return Values**

The **setjmp** subroutine returns a value of 0, unless the return is from a call to the **longjmp** function, in which case **setjmp** returns a nonzero value.

The **longjmp** subroutine cannot return 0 to the previous context. The value 0 is reserved to indicate the actual return from the **setjmp** subroutine when first called by the program. The **longjmp** subroutine does not return from where it was called, but rather, program execution continues as if the corresponding call to **setjmp** was returned with a returned value of **Value**.

If the **longjmp** subroutine is passed a **Value** parameter of 0, then execution continues as if the corresponding call to the **setjmp** subroutine had returned a value of 1. All accessible data have values as of the time the **longjmp** subroutine is called.

**Attention:** If the **longjmp** subroutine is called with a **Context** parameter that was not previously set by the **setjmp** subroutine, or if the subroutine that made the corresponding call to the **setjmp** subroutine has already returned, then the results of the **longjmp** subroutine are undefined. If the **longjmp** subroutine detects such a condition, it calls the **longjmperror** routine. If **longjmperror** returns, the program is aborted. The default version of **longjmperror** prints the message: **longjmp** or **siglongjmp** used outside of saved context to standard error and returns. Users wishing to exit in another manner can write their own version of the **longjmperror** program.

**Related Information**

The **sigsetjmp** or **siglongjmp** (**sigsetjmp or siglongjmp Subroutine” on page 226**) subroutine.

The **setlocale** subroutine

**Purpose**

Changes or queries the program’s entire current locale or portions thereof.

**Library**

Standard C Library (**libc.a**)
Syntax

```c
#include <locale.h>

char *setlocale (Category, Locale);
int Category;
const char *Locale;
```

Description

The `setlocale` subroutine selects all or part of the program's locale specified by the `Category` and `Locale` parameters. The `setlocale` subroutine then changes or queries the specified portion of the locale. The `LC_ALL` value for the `Category` parameter names the entire locale (all the categories). The other `Category` values name only a portion of the program locale.

The `Locale` parameter specifies a string that provides information needed to set certain conventions in the `Category` parameter. The components of the `Locale` parameter are language and territory. Values allowed for the locale argument are the predefined `language_territory` combinations or a user-defined locale.

If a user defines a new locale, a uniquely named locale definition source file must be provided. The character collation, character classification, monetary, numeric, time, and message information should be provided in this file. The locale definition source file is converted to a binary file by the `localedef` command. The binary locale definition file is accessed in the directory specified by the `LOCPATH` environment variable.

Note: All `setuid` and `setgid` programs will ignore the `LOCPATH` environment variable.

The default locale at program startup is the C locale. A call to the `setlocale` subroutine must be made explicitly to change this default locale environment. See [Understanding Locale Subroutines](#) in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs for `setlocale` subroutine examples.

The locale state is common to all threads within a process.

Parameters

`Category` Specifies a value representing all or part of the locale for a program. Depending on the value of the `Locale` parameter, these categories may be initiated by the values of environment variables with corresponding names. Valid values for the `Category` parameter, as defined in the `locale.h` file, are:

- `LC_ALL`: Affects the behavior of a program's entire locale.
- `LC_COLLATE`: Affects the behavior of regular expression and collation subroutines.
- `LC_CTYPE`: Affects the behavior of regular expression, character-classification, case-conversion, and wide character subroutines.
- `LC_MESSAGES`: Affects the content of messages and affirmative and negative responses.
- `LC_MONETARY`: Affects the behavior of subroutines that format monetary values.
- `LC_NUMERIC`: Affects the behavior of subroutines that format nonmonetary numeric values.
- `LC_TIME`: Affects the behavior of time-conversion subroutines.
Locale Points to a character string containing the required setting for the Category parameter.

The following are special values for the Locale parameter:

- "C" The C locale is the locale all programs inherit at program startup.
- "POSIX" Specifies the same locale as a value of "C".
- "" Specifies categories be set according to locale environment variables.
- NULL Queries the current locale environment and returns the name of the locale.

The Language Territory Table contains supported language_territory values for the Locale parameter:

<table>
<thead>
<tr>
<th>Locale Value (and Language)</th>
<th>Territory</th>
<th>Code Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ar_AA (Arabic)</td>
<td>Arabic Countries</td>
<td>IBM-1046</td>
</tr>
<tr>
<td>ar_AA (Arabic)</td>
<td>Arabic Countries</td>
<td>ISO8859-6</td>
</tr>
<tr>
<td>bg_BG (Bulgarian)</td>
<td>Bulgaria</td>
<td>ISO8859-5</td>
</tr>
<tr>
<td>cs_CZ (Czech)</td>
<td>Czech Republic</td>
<td>ISO8859-2</td>
</tr>
<tr>
<td>da_DK (Danish)</td>
<td>Denmark</td>
<td>ISO8859-1</td>
</tr>
<tr>
<td>de_CH (German)</td>
<td>Switzerland</td>
<td>ISO8859-1</td>
</tr>
<tr>
<td>de_DE (German)</td>
<td>Germany</td>
<td>ISO8859-1</td>
</tr>
<tr>
<td>el_GR (Greek)</td>
<td>Greece</td>
<td>ISO8859-7</td>
</tr>
<tr>
<td>en_GB (English)</td>
<td>Great Britain</td>
<td>ISO8859-1</td>
</tr>
<tr>
<td>en_US (English)</td>
<td>United States</td>
<td>ISO8859-1</td>
</tr>
<tr>
<td>es_ES (Spanish)</td>
<td>Spain</td>
<td>ISO8859-1</td>
</tr>
<tr>
<td>fi_FI (Finnish)</td>
<td>Finland</td>
<td>ISO8859-1</td>
</tr>
<tr>
<td>fr_BE (French)</td>
<td>Belgium</td>
<td>ISO8859-1</td>
</tr>
<tr>
<td>fr_CA (French)</td>
<td>Canada</td>
<td>ISO8859-1</td>
</tr>
<tr>
<td>fr_FR (French)</td>
<td>France</td>
<td>ISO8859-1</td>
</tr>
<tr>
<td>fr_CH (French)</td>
<td>Switzerland</td>
<td>ISO8859-1</td>
</tr>
<tr>
<td>hr_HR (Croatian)</td>
<td>Croatia</td>
<td>ISO8859-2</td>
</tr>
<tr>
<td>hu_HU (Hungarian)</td>
<td>Hungary</td>
<td>ISO8859-2</td>
</tr>
<tr>
<td>is_IS (Icelandic)</td>
<td>Iceland</td>
<td>ISO8859-1</td>
</tr>
<tr>
<td>it_IT (Italian)</td>
<td>Italy</td>
<td>ISO8859-1</td>
</tr>
<tr>
<td>iw_IL (Hebrew)</td>
<td>Israel</td>
<td>IBM-856</td>
</tr>
<tr>
<td>iw_IL (Hebrew)</td>
<td>Israel</td>
<td>ISO8859-8</td>
</tr>
<tr>
<td>Ja_JP (Japanese)</td>
<td>Japan</td>
<td>IBM-943</td>
</tr>
<tr>
<td>ja_JP (Japanese)</td>
<td>Japan</td>
<td>IBM-eucJP</td>
</tr>
<tr>
<td>ko_KR (Korean)</td>
<td>Korea</td>
<td>IBM-eucKR</td>
</tr>
<tr>
<td>mk_MK (Macedonian)</td>
<td>Former Yugoslav Republic of Macedonia</td>
<td>ISO8859-5</td>
</tr>
<tr>
<td>nl_BE (Dutch)</td>
<td>Belgium</td>
<td>ISO8859-1</td>
</tr>
<tr>
<td>nl_NL (Dutch)</td>
<td>Netherlands</td>
<td>ISO8859-1</td>
</tr>
</tbody>
</table>
Return Values
If a pointer to a string is given for the Locale parameter and the selection can be honored, the setlocale subroutine returns the string associated with the specified Category parameter for the new locale. If the selection cannot be honored, a null pointer is returned and the program locale is unchanged.

If a null is used for the Locale parameter, the setlocale subroutine returns the string associated with the Category parameter for the program's current locale. The program's locale is not changed.

A subsequent call with the string returned by the setlocale subroutine, and its associated category, will restore that part of the program locale. The string returned is not modified by the program, but can be overwritten by a subsequent call to the setlocale subroutine.

Related Information
The localeconv subroutine, nl_langinfo subroutine, rpmatch subroutine.

The localedef command.

Subroutines, Example Programs, and Libraries in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.


setpagvalue or setpagvalue64 Subroutine

Purpose
Sets the Process Authentication Group (PAG) value for a given PAG type.

Library
Security Library (libc.a)
Syntax

#include <pag.h>

int setpagvalue ( name, value )
char * name;
int value;

uint64_t setpagvalue64( name, value );
char * name;
uint64 value;

Description
The setpagvalue or setpagvalue64 subroutine sets the PAG value for a given PAG name. For these functions to succeed, the PAG name must be registered with the operating system before these subroutines are called.

Parameters

name A 1-character to 4-character, NULL-terminated name for the PAG type. Typical values include afs, dfs, pki, and krb5.
value New PAG value for the given name.

Return Values
The setpagvalue and setpagvalue64 subroutines return a PAG value upon successful completion. Upon a failure, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes
The setpagvalue and setpagvalue64 subroutines fail if the following condition is true:

EINVAL The named PAG type does not exist as part of the table.

Other errors might be set by subroutines invoked by the setpagvalue and setpagvalue64 subroutines.

Related Information

List of Security and Auditing Subroutines in AIX 5L Version 5.3 General Programming Concepts.

setpcrd Subroutine

Purpose
Sets the current process credentials.

Library
Security Library (libc.a)

Syntax
#include <usersec.h>
int setpcred (char **User, char **Credentials);
char *User;
char *Credentials;

Description
The `setpcred` subroutine sets a process’ credentials according to the `Credentials` parameter. If the `User` parameter is specified, the credentials defined for the user in the user database are used. If the `Credentials` parameter is specified, the credentials in this string are used. If both the `User` and `Credentials` parameters are specified, both the user’s and the supplied credentials are used. However, the supplied credentials of the `Credentials` parameter will override those of the user. At least one parameter must be specified.

The `setpcred` subroutine requires the `setpenv` subroutine to follow it.

Note: If the `auditwrite` subroutine is to be called from a program invoked from the `inittab` file, the `setpcred` subroutine should be called first to establish the process’ credentials.

`User` Specifies the user for whom credentials are being established.
Credentials

Defines specific credentials to be established. This parameter points to an array of null-terminated character strings that may contain the following values. The last character string must be null.

- **LOGIN_USER=%s**
  Login user name

- **REAL_USER=%s**
  Real user name

- **REAL_GROUP=%s**
  Real group name

- **GROUPS=%s**
  Supplementary group ID

- **AUDIT_CLASSES=%s**
  Audit classes

- **RLIMIT_CPU=%d**
  Process soft CPU limit

- **RLIMITFSIZE=%d**
  Process soft file size

- **RLIMIT_DATA=%d**
  Process soft data segment size

- **RLIMIT_STACK=%d**
  Process soft stack segment size

- **RLIMIT_CPU_HARD=%d**
  Process hard CPU limit

- **RLIMIT_DATA_HARD=%d**
  Process hard data segment size

- **RLIMITFSIZE_HARD=%d**
  Process hard file size

- **RLIMIT_RSS_HARD=%d**
  Process hard resident set size

- **RLIMITSTACK_HARD=%d**
  Process hard stack segment size

- **UMASK=%o**
  Process umask (file creation mask)

A process must have root user authority to set all credentials except the UMASK credential.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Hard</th>
<th>Soft</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLIMIT_CPU</td>
<td>unlimited</td>
<td>%d</td>
</tr>
<tr>
<td>RLIMIT_DATA</td>
<td>unlimited</td>
<td>%d</td>
</tr>
<tr>
<td>RLIMITFSIZE</td>
<td>%d</td>
<td>%d</td>
</tr>
<tr>
<td>RLIMIT_RSS</td>
<td>unlimited</td>
<td>%d</td>
</tr>
<tr>
<td>RLIMITSTACK</td>
<td>unlimited</td>
<td>%d</td>
</tr>
</tbody>
</table>
The soft limit credentials will override the equivalent hard limit credentials that may proceed them. To set the hard limits, the hard limit credentials should follow the soft limit credentials.

**Return Values**
Upon successful return, the `setpcred` subroutine returns a value of 0. If `setpcred` fails, a value of -1 is returned and the `errno` global variable is set to indicate the error.

**Error Codes**
The `setpcred` subroutine fails if one or more of the following are true:

- **EINVAL** The `Credentials` parameter contains invalid credentials specifications.
- **EINVAL** The `User` parameter is null and the `Credentials` parameter is either null or points to an empty string.
- **EPERM** The process does not have the proper authority to set the requested credentials.

Other errors may be set by subroutines invoked by the `setpcred` subroutine.

**Related Information**
The `auditwrite`, `ckuseracct`, `ckuserID`, `getpcred`, `getenv` subroutine, `setpcred` subroutine, `setenv` subroutine.

List of Security and Auditing Subroutines [Subroutines Overview] in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

**setenv Subroutine**

**Purpose**
Sets the current process environment.

**Library**
Security Library (libc.a)

**Syntax**
```
#include <usersec.h>

int setenv (char *User, int Mode, char **Environment, char *Command);
```

**Description**
The `setenv` subroutine first sets the environment of the current process according to its parameter values, and then sets the working directory and runs a specified command. If the `User` parameter is specified, the process environment is set to that of the specified user, the user's working directory is set, and the specified command run. If the `User` parameter is not specified, then the environment and working directory are set to that of the current process, and the command is run from this process. The environment consists of both user-state and system-state environment variables.

**Note:** The `setenv` subroutine requires the `setpcred` subroutine to precede it.
The **setpenv** subroutine performs the following steps:

### Setting the Process Environment

The first step involves changing the user-state and system-state environment. Since this is dependent on the values of the **Mode** and **Environment** parameters, see the description for the **Mode** parameter for more information.

### Setting the Process Current Working Directory

After the user-state and system-state environment is set, the working directory of the process may be set. If the **Mode** parameter includes the **PENV_INIT** value, the current working directory is changed to the user's initial login directory (defined in the `/etc/passwd` file). Otherwise, the current working directory is unchanged.

### Executing the Initial Program

After the working directory of the process is reset, the initial program (usually the shell interpreter) is executed. If the **Command** parameter is null, the shell from the user database is used. If the parameter is not defined, the shell from the user-state environment is used and the **Command** parameter defaults to the `/usr/bin/sh` file. If the **Command** parameter is not null, it specifies the command to be executed. If the **Mode** parameter contains the **PENV_ARGV** value, the **Command** parameter is assumed to be in the **argv** structure and is passed to the **execve** subroutine. The string contained in the **Command** parameter is used as the **Path** parameter of the **execve** subroutine. If the **Mode** parameter does not contain **PENV_ARGV** value, the **Command** parameter is parsed into an **argv** structure and executed. If the **Command** parameter contains the **$SHELL** value, substitution is done prior to execution.

**Note:** This step will fail if the **Command** parameter contains the **$SHELL** value but the user-state environment does not contain the **SHELL** value.

### Parameters

**Command**

Specifies the command to be executed. If the **Mode** parameter contains the **PENV_ARGV** value, then the **Command** parameter is assumed to be a valid argument vector for the **execve** subroutine.

**Environment**

Specifies the value of user-state and system-state environment variables in the same format returned by the **getpenv** subroutine. The user-state variables are prefaced by the keyword **USRENVIRON**:, and the system-state variables are prefaced by the keyword **SYSENVIRON**:. Each variable is defined by a string of the form **var=value**, which is an array of null-terminated character pointers.

**Mode**

Specifies how the **setpenv** subroutine is to set the environment and run the command. This parameter is a bit mask and must contain only one of the following values, which are defined in the **usersec.h** file:

**PENV_INIT**

The user-state environment is initialized as follows:

**AUTHSTATE**

Retained from the current environment. If the **AUTHSTATE** value is not present, it is defaulted to the **compat** value.
KRB5CCNAME
Retained from the current environment. This value is defined if you authenticated through the Distributed Computing Environment (DCE).

USER
Set to the name specified by the User parameter or to the name corresponding to the current real user ID. The name is shortened to a maximum of PW_USERNAME_LEN, including the trailing NUL character.

PW_USERNAME_LEN is the running system’s maximum value. The value of PW_USERNAME_LEN can be at the most MAXIMPL_LOGIN_NAME_MAX (or 256 characters), and must be at least 9 characters.

LOGIN
Set to the name specified by the User parameter or to the name corresponding to the current real user ID. If set by the User parameter, this value is the complete login name, which may include a DCE cell name.

LOGNAME
Set to the current system environment variable LOGNAME.

TERM
Retained from the current environment. If the TERM value is not present, it is defaulted to an IBM6155.

SHELL
Set from the initial program defined for the real user ID of the current process. If no program is defined, then the /usr/bin/sh shell is used as the default.

HOME
Set from the home directory defined for the real user ID of the current process. If no home directory is defined, the default is /home/guest.

PATH
Set initially to the value for the PATH value in the /etc/environment file. If not set, it is destructively replaced by the default value of PATH=/usr/bin:$HOME:. (The final period specifies the working directory). The PATH variable is destructively replaced by the usrenv attribute for this user in the /etc/security/environ file if the PATH value exists in the /etc/environment file.

The following files are read for additional environment variables:

/etc/environment
Variables defined in this file are added to the environment.

/etc/security/environ
Environment variables defined for the user in this file are added to the user-state environment.

The user-state variables in the Environment parameter are added to the user-state environment. These are preceded by the USRENVIRON: keyword.

The system-state environment is initialized as follows:

LOGNAME
Set to the current LOGNAME value in the protected user environment. The login (tsm) command passes this value to the setpenv subroutine to ensure correctness.

NAME
Set to the login name corresponding to the real user ID.

TTY
Set to the TTY name corresponding to standard input.

The following file is read for additional environment variables:

/etc/security/environ
The system-state environment variables defined for the user in this file are added to the environment. The system-state variables in the Environment parameter are added to the environment. These are preceded by the SYSENVIRON keyword.
The existing user-state and system-state environment variables are preserved and the variables defined in the Environment parameter are added.

The existing environment is cleared and totally replaced by the content of the Environment parameter.

Closes all open file descriptors, except 0, 1, and 2, before executing the command. This value must be logically ORed with PENV_DELTA, PENV_RESET, or PENV_INIT. It cannot be used alone.

The new shell will not be treated as a login shell. Only valid when used with the PENV_INIT flag.

For both system-state and user-state environments, variable substitution is performed.

The Mode parameter may also contain:

Specifies that the Command parameter is already in argv format and need not be parsed. This value must be logically ORed with PENV_DELTA, PENV_RESET, or PENV_INIT. It cannot be used alone.

Specifies the user name whose environment and working directory is to be set and the specified command run. If a null pointer is given, the current real uid is used to determine the name of the user.

If the environment was successfully established, this function does not return. If the setpenv subroutine fails, a value of -1 is returned and the errno global variable is set to indicate the error.

The setpenv subroutine fails if one or more of the following are true:

The Mode parameter contains values other than PENV_INIT, PENV_DELTA, PENV_RESET, or PENV_ARGV.

The Mode parameter contains more than one of PENV_INIT, PENV_DELTA, or PENV_RESET values.

The Environment parameter is neither null nor empty, and does not contain a valid environment string.

The caller does not have read access to the environment defined for the system, or the user does not have permission to change the specified attributes.

Other errors may be set by subroutines invoked by the setpenv subroutine.

The executables, execve, execle, execvp, or exec subroutine, getpenv subroutine, setpcred subroutine.

The login command, su command.
setpgid or setpgrp Subroutine

Purpose
Sets the process group ID.

Libraries
setpgid: Standard C Library (libc.a)
setpgrp: Standard C Library (libc.a); Berkeley Compatibility Library (libbsd.a)

Syntax
#include <unistd.h>

pid_t setpgid (ProcessID, ProcessGroupID)

pid_t ProcessID, ProcessGroupID;

pid_t setpgrp ( )

Description
The setpgid subroutine is used either to join an existing process group or to create a new process group within the session of the calling process. The process group ID of a session leader does not change. Upon return, the process group ID of the process having a process ID that matches the ProcessID value is set to the ProcessGroupID value. As a special case, if the ProcessID value is 0, the process ID of the calling process is used. If ProcessGroupID value is 0, the process ID of the indicated process is used.

This function is implemented to support job control.

The setpgrp subroutine in the libc.a library supports a subset of the function of the setpgid subroutine. It has no parameters. It sets the process group ID of the calling process to be the same as its process ID and returns the new value.

In BSD systems, the setpgrp subroutine is defined with two parameters, as follows:

pid_t setpgrp (ProcessID, ProcessGroup)

pid_t ProcessID, ProcessGroup;

Parameters
ProcessID Specifies the process whose process group ID is to be changed.
ProcessGroupID Specifies the new value of calling process group ID.

Return Values
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the errno global variable is set to indicate the error.
**Error Codes**
The `setpgid` subroutine is unsuccessful if one or more of the following is true:

- **EACCES** The value of the `ProcessID` parameter matches the process ID of a child process of the calling process and the child process has successfully executed one of the `exec` subroutines.
- **EINVAL** The value of the `ProcessGroupID` parameter is less than 0, or is not a valid value.
- **ENOSYS** The `setpgid` subroutine is not supported by this implementation.
- **EPERM** The process indicated by the value of the `ProcessID` parameter is a session leader.
- **EPERM** The value of the `ProcessID` parameter 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.
- **EPERM** The value of the `ProcessGroupID` parameter is valid, but does not match the process ID of the process indicated by the `ProcessID` parameter. There is no process with a process group ID that matches the value of the `ProcessGroupID` parameter in the same session as the calling process.
- **ESRCH** The value of the `ProcessID` parameter does not match the process ID of the calling process of a child process of the calling process.

**Related Information**
The `getpid` subroutine.

---

**setpri Subroutine**

**Purpose**
Sets a process scheduling priority to a constant value.

**Library**
Standard C Library (`libc.a`)

**Syntax**
```
#include <sys/sched.h>

int setpri (pid_t ProcessID, int Priority);
```

**Description**
The `setpri` subroutine sets the scheduling priority of all threads in a process to be a constant. All threads have their scheduling policies changed to `SCHED_RR`. A process nice value and CPU usage can no longer be used to determine a process scheduling priority. Only processes that have root user authority can set a process scheduling priority to a constant.

**Parameters**
- **ProcessID** Specifies the process ID. If this value is 0 then the current process scheduling priority is set to a constant.
- **Priority** Specifies the scheduling priority for the process. A lower number value designates a higher scheduling priority. The `Priority` parameter must be in the range `PRIORITY_MIN` < `Priority` < `PRIORITY_MAX`. (See the `sys/sched.h` file.)

**Return Values**
Upon successful completion, the `setpri` subroutine returns the former scheduling priority of the process just changed. Otherwise, a value of -1 is returned and the `errno` global variable is set to indicate the error.
Error Codes
The `setpri` subroutine is unsuccessful if one or more of the following is true:

- **EINVAL** The priority specified by the `Priority` parameter is outside the range of acceptable priorities.
- **EPERM** The process executing the `setpri` subroutine call does not have root user authority.
- **ESRCH** No process can be found corresponding to that specified by the `ProcessID` parameter.

Related Information
The `getpri` subroutine.

Performance-Related Subroutines in *AIX 5L Version 5.3 Performance Management Guide.*

**setpwdb or endpwdb Subroutine**

**Purpose**
Opens or closes the authentication database.

**Library**
Security Library (`libc.a`)

**Syntax**
```
#include <userpw.h>

int setpwdb (Mode);
int Mode;
int endpwdb ( )
```

**Description**
These functions are used to open and close access to the authentication database. Programs that call either the `getuserpw` or `putuserpw` subroutine should call the `setpwdb` subroutine to open the database and the `endpwdb` subroutine to close the database.

The `setpwdb` subroutine opens the authentication database in the specified mode, if it is not already open. The open count is increased by 1.

The `endpwdb` subroutine decreases the open count by one and closes the authentication database when this count drops to 0. Subsequent references to individual data items can cause a memory access violation. The `endpwdb` subroutine also frees the space that was allocated by either the `getuserpw`, `putuserpw`, or `putuserpwhist` subroutine. For security reasons, freeing the space clears the password field. Any uncommitted changed data is lost.

**Parameters**
- **Mode** Specifies the mode of the open. This parameter may contain one or more of the following values, defined in the `usersec.h` file:
  - **S_READ** Specifies read access.
  - **S_WRITE** Specifies update access.
Return Values
The setpwdb and endpwdb subroutines return a value of 0 to indicate success. Otherwise, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes
The setpwdb and endpwdb subroutines fail if the following is true:

EACCES     Access permission is denied for the data request.

Both of these functions return errors from other subroutines.

Security
Access Control: The calling process must have access to the authentication data.

Files Accessed:

<table>
<thead>
<tr>
<th>Modes</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>/etc/security/passwd</td>
</tr>
<tr>
<td>rw</td>
<td>/etc/passwd</td>
</tr>
</tbody>
</table>

Related Information
The getgroupattr subroutine, getuserattr subroutine, getuserpw, putuserpw, or putuserpwhist subroutine.

setroledb or endroledb Subroutine

Purpose
Opens and closes the role database.

Library
Security Library (libc.a)

Syntax
```c
#include <usersec.h>
int setroledb(Mode)
int Mode;
int endroledb
```

Description
These functions may be used to open and close access to the role database. Programs that call the getroleattr subroutine should call the setroledb subroutine to open the role database and the endroledb subroutine to close the role database.

The setroledb subroutine opens the role database in the specified mode, if it is not already open. The open count is increased by 1.
The `endroledb` subroutine decreases the open count by 1 and closes the role database when this count goes to 0. Any uncommitted changed data is lost.

**Parameters**

*Mode* Specifies the mode of the open. This parameter may contain one or more of the following values defined in the `usersec.h` file:

- `S_READ` Specifies read access.
- `S_WRITE` Specifies update access.

**Return Values**

The `setroledb` and `endroledb` subroutines return a value of 0 to indicate success. Otherwise, a value of -1 is returned and the `errno` global variable is set to indicate the error.

**Error Codes**

The `setroledb` subroutine fails if the following is true:

- `EACCES` Access permission is denied for the data request.

Both subroutines return errors from other subroutines.

**Security**

Files Accessed: The calling process must have access to the role data.

Mode File `rw/etc/security/roles`

**Related Information**

The `getroleattr`, `nextrole`, or `putroleattr` subroutine.

---

**setsid Subroutine**

**Purpose**

Creates a session and sets the process group ID.

**Library**

Standard C Library (`libc.a`)

**Syntax**

```c
#include <unistd.h>

pid_t setsid (void)
```

**Description**

The `setsid` subroutine creates a new session if the calling process is not a process group leader. Upon return, the calling process is the session leader of this new session, the process group leader of a new process group, and has no controlling terminal. The process group ID of the calling process is set equal to its process ID. The calling process is the only process in the new process group and the only process in the new session.
**Return Values**
Upon successful completion, the value of the new process group ID is returned. Otherwise, \((\text{pid\_t})\) -1 is returned and the \texttt{errno} global variable is set to indicate the error.

**Error Codes**
The \texttt{setsid} subroutine is unsuccessful if the following is true:

- \texttt{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.

**Related Information**
The \texttt{fork} subroutine, \texttt{setgid}, \texttt{getpgrp}, or \texttt{setpgid} subroutine, \texttt{setpgid} subroutine, \texttt{setpgrp} subroutine.

**setuid, setruid, seteuid, setreuid or setuidx Subroutine**

**Purpose**
Sets the process user IDs.

**Library**
Standard C Library (\texttt{libc.a})

**Syntax**

```c
#include <unistd.h>

int setuid (\texttt{UID});

uid_t \texttt{UID};

int setruid (\texttt{RUID});

uid_t \texttt{RUID};

int seteuid (\texttt{EUID});

uid_t \texttt{EUID};

int setreuid (\texttt{RUID, EUID});

uid_t \texttt{RUID};

uid_t \texttt{EUID};

#include <unistd.h>
#include <sys/id.h>

int setuidx (\texttt{which}, \texttt{UID});

int \texttt{which};

int \texttt{UID};
```

**Description**
The \texttt{setuid}, \texttt{setruid}, \texttt{seteuid}, and \texttt{setreuid} subroutines reset the process user IDs. The following semantics are supported:

- \texttt{setuid} If the effective user ID of the process is the root user, the process’s real, effective, and saved user IDs are set to the value of the \texttt{UID} parameter. Otherwise, the process effective user ID is reset if the \texttt{UID} parameter specifies either the current real or saved user IDs.

- \texttt{seteuid} The process effective user ID is reset if the \texttt{UID} parameter is equal to either the current real or saved user IDs or if the effective user ID of the process is the root user.

- \texttt{setruid} The \texttt{EPERM} error code is always returned. Processes cannot reset only their real user IDs.
setreuid  The RUID and EUID parameters can have the following two possibilities:

\[
\text{RUID} \neq \text{EUID}
\]
If the EUID parameter specifies either the process's real or saved user IDs, the process effective user ID is set to the EUID parameter. Otherwise, the EPERM error code is returned.

\[
\text{RUID} = \text{EUID}
\]
If the process effective user ID is the root user, the process's real and effective user IDs are set to the EUID parameter. Otherwise, the EPERM error code is returned.

If both the real user ID and effective user ID are changed, the saved user ID is set to the new effective user ID. Note that this change results in a loss of original privileges.

setuidx  The which parameter can have one of the following values:

\[
\text{ID\_EFFECTIVE}
\]
UID must be either the real or saved UID. The effective user ID for the current process will be set to UID.

\[
\text{ID\_EFFECTIVE\_REAL}
\]
Invoker must have appropriate privilege. The real and effective user ID for the current process will be set to UID.

\[
\text{ID\_EFFECTIVE\_REAL\_SAVED}
\]
Invoker must have appropriate privilege. The real, effective and saved user ID for the current process will be set to UID.

\[
\text{ID\_LOGIN}
\]
Invoker must have appropriate privilege. The login UID for the current process will be set to UID.

The real and effective user ID parameters can have a value of -1. If the value is -1, the actual value for the UID parameter is set to the corresponding current the UID parameter of the process.

The operating system does not support setuid or setgid ("setgid, setrgid, setegid, setregid, or setgidx Subroutine" on page 167) shell scripts.

These subroutines are part of Base Operating System (BOS) Runtime.

Parameters

\[
\text{UID}
\]
Specifies the user ID to set.

\[
\text{EUID}
\]
Specifies the effective user ID to set.

\[
\text{RUID}
\]
Specifies the real user ID to set.

\[
\text{which}
\]
Specifies which user ID values to set.

Return Values

Upon successful completion, the setuid, seteuid, setreuid, and setuidx subroutines return a value of 0. Otherwise, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes

The setuid, seteuid, setreuid, and setuidx subroutines are unsuccessful if either of the following is true:

\[
\text{EINVAL}
\]
The value of the UID or EUID parameter is not valid.

\[
\text{EPERM}
\]
The process does not have the appropriate privileges and the UID and EUID parameters are not equal to either the real or saved user IDs of the process.
setuserdb or enduserdb Subroutine

Purpose

Opens and closes the user database.

Library

Security Library (libc.a)

Syntax

```c
#include <usersec.h>

int setuserdb (Mode);
int Mode;
int enduserdb ( )
```

Description

These functions may be used to open and close access to the user database. Programs that call either the getuserattr or getgroupattr subroutine should call the setuserdb subroutine to open the user database and the enduserdb subroutine to close the user database.

The setuserdb subroutine opens the user database in the specified mode, if it is not already open. The open count is increased by 1.

The enduserdb subroutine decreases the open count by 1 and closes the user database when this count goes to 0. Any uncommitted changed data is lost.

Parameters

`Mode` Specifies the mode of the open. This parameter may contain one or more of the following values defined in the usersec.h file:

- **S_READ**
  Specifies read access

- **S_WRITE**
  Specifies update access.

Return Values

The setuserdb and enduserdb subroutines return a value of 0 to indicate success. Otherwise, a value of -1 is returned and the errno global variable is set to indicate the error.
Error Codes

The setuserdb subroutine fails if the following is true:

EACCES Access permission is denied for the data request.

Both subroutines return errors from other subroutines.

Security

Files Accessed: The calling process must have access to the user data. Depending on the actual attributes accessed, this may include:

<table>
<thead>
<tr>
<th>Modes</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>/etc/passwd</td>
</tr>
<tr>
<td>rw</td>
<td>/etc/group</td>
</tr>
<tr>
<td>rw</td>
<td>/etc/security/user</td>
</tr>
<tr>
<td>rw</td>
<td>/etc/security/limits</td>
</tr>
<tr>
<td>rw</td>
<td>/etc/security/group</td>
</tr>
<tr>
<td>rw</td>
<td>/etc/security/environ</td>
</tr>
</tbody>
</table>

Related Information

The setgroupattr subroutine, getgroupattr subroutine, getuserattr subroutine, getuserpw subroutine, setpwdb subroutine, "setpwdb or endpwdb Subroutine" on page 184 subroutine.

List of Security and Auditing Subroutines and Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

sgetl or sputl Subroutine

Purpose

Accesses long numeric data in a machine-independent fashion.

Library

Object File Access Routine Library (libld.a)

Syntax

long sgetl (Buffer)
char *Buffer;

void sputl (Value, Buffer)
long Value;
char *Buffer;

Description

The sgetl subroutine retrieves four bytes from memory starting at the location pointed to by the Buffer parameter. It then returns the bytes as a long Value with the byte ordering of the host machine.

The sputl subroutine stores the four bytes of the Value parameter into memory starting at the location pointed to by the Buffer parameter. The order of the bytes is the same across all machines.
Using the `sputl` and `sgetl` subroutines together provides a machine-independent way of storing long numeric data in an ASCII file. For example, the numeric data stored in the portable archive file format can be accessed with the `sputl` and `sgetl` subroutines.

**Parameters**

<table>
<thead>
<tr>
<th>Value</th>
<th>Specifies a 4-byte value to store into memory.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer</td>
<td>Points to a location in memory.</td>
</tr>
</tbody>
</table>

**Related Information**

- The `ar` command, `dump` command.
- The `ar` file format, `a.out` file format.

Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

### shm_open Subroutine

**Purpose**

Opens a shared memory object.

**Library**

Standard C Library (`libc.a`)

**Syntax**

```c
#include <sys/mman.h>

int shm_open (const char *name, int oflag, mode_t mode);
```

**Description**

The `shm_open` subroutine establishes a connection between a shared memory object and a file descriptor. It creates an open file description that refers to the shared memory object and a file descriptor that refers to that open file description. This file descriptor is used by other subroutines to refer to that shared memory object.

The `name` parameter points to a string naming a shared memory object. The `name` parameter does not appear in the file system and is not visible to other subroutines that take pathnames as arguments. The `name` parameter must conform to the construction rules for a pathname.

If successful, the `shm_open` subroutine returns a file descriptor for the shared memory object that is the lowest numbered 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 processes. The `FD_CLOEXEC` file descriptor flag associated with the new file descriptor is set.

The file status flags and file access modes of the open file description are according to the value of the `oflag` parameter. The `oflag` parameter is the bitwise-inclusive OR of the following flags defined in the `fcntl.h` header file.
Parameters

name Points to a string naming a shared memory object.
oflag Specifies the flags to be used by the shm_open subroutine.
mode Sets the value of the permission bits of the shared memory object.

Read-Write Flags
Applications specify exactly one of the first two values (access modes) below in the value of the oflag parameter:

- **O_RDONLY** Open for read access only.
- **O_RDWR** Open for read or write access.

Other Flags
Any combination of the remaining flags may be specified in the value of the oflag parameter:

- **O_CREAT** If the shared memory object exists, this flag has no effect, except as noted under the O_EXCL flag below. Otherwise, the shared memory object is created, the user ID of the shared memory object is set to the effective user ID of the process, and the group ID of the shared memory object is set to the effective group ID of the process. The permission bits of the shared memory object are set to the value of the mode parameter except those set in the file mode creation mask of the process. Only the low-order 9 bits of the mode parameter are taken into account. The shared memory object has a size of zero.
- **O_EXCL** If the O_EXCL and O_CREAT flags are set, the shm_open subroutine fails if the shared memory object exists. The O_EXCL flag is ignored if the O_CREAT flag is not set.
- **O_TRUNC** If the shared memory object exists and it is successfully opened, the O_RDWR flag, the object is truncated to zero length, and the mode and owner is unchanged by the shm_open call.

Return Values
Upon successful completion, the shm_open subroutine returns a non-negative integer representing the lowest numbered unused file descriptor. If unsuccessful, it returns -1 and sets errno to indicate the error.

Error Codes

- **EACCESS** The shared memory object exists and the permissions specified by the oflag parameter are denied, or the shared memory object does not exist and permission to create the shared memory object is denied, or the O_TRUNC flag is specified and write permission is denied.
- **EEXIST** The O_CREAT and O_EXCL flags are set and the named shared memory object already exists.
- **EINVAL** The shm_open subroutine is not supported for an empty name string, or the name parameter is missing, or the oflag parameter contains an invalid value.
- **EFAULT** The name parameter points outside of the allocated address space of the process.
- **EMFILE** Too many file descriptors are currently in use by this process.
- **ENAMETOOLONG** The length of the name parameter exceeds PATH_MAX or a pathname component is longer than NAME_MAX.
- **ENFILE** Too many shared memory objects are currently open in the system.
- **ENOENT** The O_CREAT flag is not set and the named shared memory object does not exist.
- **ENOMEM** The system is unable to allocate resources.
- **ENOTSUP** This function is not supported with processes that have been checkpoint-restart’ed.
- **ENOSPC** There is insufficient space for the creation of the new shared memory object.
Related Information

“shmat Subroutine” on page 194, “shmctl Subroutine” on page 197, “shmdt Subroutine” on page 200, and “shm_unlink Subroutine.”

close, dup, exec, and mmap in AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 1

umask Command in AIX 5L Version 5.3 Commands Reference, Volume 5.

fcntl.h File in AIX 5L Version 5.3 Files Reference.

shm_unlink Subroutine

Purpose

Removes a shared memory object.

Library

Standard C Library (libc.a)

Syntax

#include <sys/mman.h>

int shm_unlink (name);
const char *name;

Description

The shm_unlink subroutine removes the name of the shared memory object named by the string pointed to by the name parameter.

If one or more references to the shared memory object exist when the object is unlinked, the name is removed before the shm_unlink subroutine returns, but the removal of the memory object contents is postponed until all open and map references to the shared memory object have been removed.

Even if the object continues to exist after the last shm_unlink call, reuse of the name subsequently causes the shm_open subroutine to behave as if no shared memory object of this name exists. In other words, the shm_open subroutine will fail if O_CREAT is not set, or will create a new shared memory object if O_CREAT is set.

Parameters

name Specifies the name of the shared memory object to be unlinked.

Return Values

Upon successful completion, zero is returned. Otherwise, -1 is returned and errno is set to indicate the error. If -1 is returned, the named shared memory object is not changed by the subroutine call.

Error Codes

The shm_unlink subroutine fails if:

EACCES Permission is denied to unlink the named shared memory object.
EFAULT The name parameter points outside of the allocated address space of the process.
shmat Subroutine

Purpose
Attaches a shared memory segment or a mapped file to the current process.

Library
Standard C Library (libc.a)

Syntax
#include <sys/shm.h>

void *shmat (SharedMemoryID, SharedMemoryAddress, SharedMemoryFlag)
int SharedMemoryID, SharedMemoryFlag;
const void *SharedMemoryAddress;

Description
The shmat subroutine attaches the shared memory segment or mapped file specified by the SharedMemoryID parameter (returned by the shmget subroutine), or file descriptor specified by the SharedMemoryID parameter (returned by the open subroutine) to the address space of the calling process.

A call to the shmat subroutine on a file descriptor that identifies an open shared memory object fails with EINVAL.

To learn more about the limits that apply to shared memory, see the Inter-Process Communication (IPC) Limits article in AIX 5L Version 5.3 General Programming Concepts.

Note: The following applies to AIX 4.2.1 and later releases for 32-bit processes only.

An extended shmat capability is available. If an environment variable EXTSHM=ON is defined then processes executing in that environment will be able to create and attach more than eleven shared memory segments.

The segments can be of size from 1 byte to 2GB. The process can attach segments larger than 256MB into the address space for the size of the segment. Another segment could be attached at the end of the first one in the same 256MB segment region. The address at which a process can attach is at page boundaries - a multiple of SHMLBA_EXTSHM bytes. For segments larger than 256MB in size, if EXTSHM=ON is not defined, the address at which a process can attach is at 256MB boundaries, which is a multiple of SHMLBA bytes.
The segments can be of size from 1 byte to 256MB. The process can attach these segments into the address space for the size of the segment. Another segment could be attached at the end of the first one in the same 256MB segment region. The address at which a process can attach will be at page boundaries - a multiple of SHMLBA_EXTSHM bytes.

The maximum address space available for shared memory with or without the environment variable and for memory mapping is 2.75GB. An additional segment register "0xE" is available so that the address space is from 0x30000000 to 0xE0000000. However, a 256MB region starting from 0xD0000000 will be used by the shared libraries and is therefore unavailable for shared memory regions or mmaped regions.

On AIX 5.2 and later, a 32-bit process running with the very large address space model has up to 3.25 GB of address space available for the shmat and mmap memory mappings. For a 32-bit process with the very large address space model, the address space available for mappings is from 0x30000000 to 0xFFFFFFFF. This extended address range applies to both extended shmat and standard shmat. For more information on how to use the very large address space model, see the Understanding the Very Large Address-Space Model article in AIX 5L Version 5.3 General Programming Concepts.

There are some restrictions on the use of the extended shmat feature. These shared memory regions can not be used as I/O buffers where the unpinning of the buffer occurs in an interrupt handler. The restrictions on the use are the same as that of mmap buffers.

The smaller region sizes are not supported for mapping files. Regardless of whether EXTSHM=ON or not, mapping a file will consume at least 256MB of address space.

The SHM_SIZE shmctl command is not supported for segments created with EXTSHM=ON.

A segment created with EXTSHM=ON can be attached by a process without EXTSHM=ON. This will consume an area of address space that is a multiple of 256MB in size, regardless of the size of the shared memory region.

A segment created without EXTSHM=ON can be attached by a process with EXTSHM=ON. This will consume an area of address space that is a multiple of 256MB in size, regardless of the size of the shared memory region.

The environment variable provides the option of executing an application either with the additional functionality of attaching more than 11 segments when EXTSHM=ON, or the higher-performance access to 11 or fewer segments when the environment variable is not set.

The EXTSHM environment variable supports two additional values, EXTSHM=1SEG and EXTSHM=MSEG. All three options let users create more than 11 segments.

The EXTSHM=1SEG option defaults to the same behavior as EXTSHM=ON, which is to make memory mapped segments (type MMAP) of shared memories less than 256 MB, and SHMAT’ed segments (type WORKING) of shared memories greater than or equal to 256 MB. The EXTSHM=MSEG option creates memory mapped segments of all shared memories, regardless of size. This option provides better use of memory space.

Parameters

SharedMemoryID Specifies an identifier for the shared memory segment.
Identifies the segment or file attached at the address specified by the `SharedMemoryAddress` parameter, as follows:

- **SHM_RND** flag is set in the `SharedMemoryFlag` parameter, the segment or file is attached at the next lower segment boundary. This address is given by `(SharedMemoryAddress - (SharedMemoryAddress modulo SHMLBA_EXTSHM if environment variable EXTSHM=ON or SHMLBA if not)).` SHMLBA specifies the low boundary address multiple of a segment.

- **SHM_RND** flag is not set in the `SharedMemoryFlag` parameter, the segment or file is attached at the address given by the `SharedMemoryAddress` parameter. If this address does not point to a SHMLBA_EXTSHM boundary if the environment variable EXTSHM=ON or SHMLBA boundary if not, the `shmat` subroutine returns the value -1 and sets the `errno` global variable to the `EINVAL` error code. SHMLBA specifies the low boundary address multiple of a segment.

**SharedMemoryFlag**

Specify several options. Its value is either 0 or is constructed by logically ORing one or more of the following values:

- **SHM_COPY**
  - Changes an open file to deferred update (see the `openx` subroutine). Included only for compatibility with previous versions of the operating system.

- **SHM_MAP**
  - Maps a file onto the address space instead of a shared memory segment. The `SharedMemoryID` parameter must specify an open file descriptor in this case.

- **SHM_RDONLY**
  - Specifies read-only mode instead of the default read-write mode.

- **SHM_RND**
  - Rounds the address given by the `SharedMemoryAddress` parameter to the next lower segment boundary, if necessary.

The `shmat` subroutine makes a shared memory segment addressable by the current process. The segment is attached for reading if the **SHM_RDONLY** flag is set and the current process has read permission. If the **SHM_RDONLY** flag is not set and the current process has both read and write permission, it is attached for reading and writing.

If the **SHM_MAP** flag is set, file mapping takes place. In this case, the `shmat` subroutine maps the file open on the file descriptor specified by the `SharedMemoryID` onto a segment. The file must be a regular file. The segment is then mapped into the address space of the process. A file of any size can be mapped if there is enough space in the user address space.

When file mapping is requested, the `SharedMemoryFlag` parameter specifies how the file should be mapped. If the **SHM_RDONLY** flag is set, the file is mapped read-only. To map read-write, the file must have been opened for writing.

All processes that map the same file read-only or read-write map to the same segment. This segment remains mapped until the last process mapping the file closes it.

A mapped file opened with the **O_DEFER** update has deferred update. That is, changes to the shared segment do not affect the contents of the file resident in the file system until an `fsync` subroutine is issued to the file descriptor for which the mapping was requested. Setting the **SHM_COPY** flag changes the file to the deferred state. The file remains in this state until all processes close it. The **SHM_COPY** flag is provided only for compatibility with Version 2 of the operating system. New programs should use the **O_DEFER** open flag.

A file descriptor can be used to map the corresponding file only once. To map a file several times requires multiple file descriptors.
When a file is mapped onto a segment, the file is referenced by accessing the segment. The memory paging system automatically takes care of the physical I/O. References beyond the end of the file cause the file to be extended in page-sized increments. The file cannot be extended beyond the next segment boundary.

**Attention:** When a file is mapped, use of standard file system calls, such as `truncate` and `write`, are discouraged and might produce unexpected results, especially in a multithreaded environment. In particular, the `write` system call, upon completion, sets the size to the new end-of-file. Any `shmat` changes that occur concurrently past this new end-of-file might be lost. Concurrent change of the mapped region and use of the `write` system call are highly discouraged.

**Return Values**

When successful, the segment start address of the attached shared memory segment or mapped file is returned. Otherwise, the shared memory segment is not attached, the `errno` global variable is set to indicate the error, and a value of -1 is returned.

**Error Codes**

The `shmat` subroutine is unsuccessful and the shared memory segment or mapped file is not attached if one or more of the following are true:

- **EACCES** The calling process is denied permission for the specified operation.
- **EAGAIN** The file to be mapped has enforced locking enabled, and the file is currently locked.
- **EBADF** A file descriptor to map does not refer to an open regular file.
- **EEXIST** The file to be mapped has already been mapped.
- **EINVAL** The `SHM_RDONLY` and `SHM_COPY` flags are both set.
- **EINVAL** The `shmat` subroutine was used with a file descriptor obtained from a call to the `shm_open` subroutine.
- **EINVAL** The `SharedMemoryID` parameter is not a valid shared memory identifier.
- **EINVAL** The `SharedMemoryAddress` parameter is not equal to 0, and the value of `(SharedMemoryAddress - (SharedMemoryAddress modulo SHMLBA_EXTSHM if the environment variable EXTSHM=ON or SHMLBA if not ) ) points outside the address space of the process.
- **EINVAL** The `SharedMemoryAddress` parameter is not equal to 0, the `SHM_RND` flag is not set in the `SharedMemoryFlag` parameter, and the `SharedMemoryAddress` parameter points to a location outside of the address space of the process.
- **EMFILE** The number of shared memory segments attached to the calling process exceeds the system-imposed limit.
- **ENOMEM** The available data space in memory is not large enough to hold the shared memory segment. ENOMEM is always returned if a 32-bit process tries to attach a shared memory segment larger than 2GB.
- **ENOMEM** The available data space in memory is not large enough to hold the mapped file data structure.

**Related Information**

The `exec`, `exit`, `fclear`, `fork`, `fsync`, `mmap`, `munmap`, `open`, `truncate`, `readv`, `shmdt`, `shmget`, `writev`, `any other subroutine, subroutine, function, or command.

The `ipcs` command and `ipcrm` command.

**List of Memory Manipulation Services** | **Subroutines Overview** | **Understanding Memory Mapping** in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

**shmctl Subroutine**

**Purpose**

Controls shared memory operations.
Library
Standard C Library (libc.a)

Syntax
#include <sys/shm.h>

int shmctl (SharedMemoryID, Command, Buffer)
int SharedMemoryID, Command;
struct shmid_ds * Buffer;

Description
The shmctl subroutine performs a variety of shared-memory control operations as specified by the Command parameter.

The following limits apply to shared memory:
• Maximum shared-memory segment size is:
  – 256M bytes before AIX 4.3.1
  – 2G bytes for AIX 4.3.1 through AIX 5.1
  – 64G bytes for 64-bit applications for AIX 5.1 and later
• Minimum shared-memory segment size is 1 byte.
• Maximum number of shared memory IDs is 4096 for operating system releases before AIX 4.3.2 and 131072 for AIX 4.3.2 and following.

Parameters

SharedMemoryID Specifies an identifier returned by the shmget subroutine.
Buffer Indicates a pointer to the shmid_ds structure. The shmid_ds structure is defined in the sys/shm.h file.
The following commands are available:

**IPC_STAT**
Obtains status information about the shared memory segment identified by the `SharedMemoryID` parameter. This information is stored in the area pointed to by the `Buffer` parameter. The calling process must have read permission to run this command.

**IPC_SET**
Sets the user and group IDs of the owner as well as the access permissions for the shared memory segment identified by the `SharedMemoryID` parameter. This command sets the following fields:

```
shm_perm.uid  /* owning user ID */
shm_perm.gid  /* owning group ID */
shm_perm.mode /* permission bits only */
```

You must have an effective user ID equal to root or to the value of the `shm_perm.uid` or `shm_perm.cuid` field in the `shm_id_ds` data structure identified by the `SharedMemoryID` parameter.

**IPC_RMID**
Removes the shared memory identifier specified by the `SharedMemoryID` parameter from the system and erases the shared memory segment and data structure associated with it. This command is only executed by a process that has an effective user ID equal either to that of `superuser` or to the value of the `shm_perm.uid` or `shm_perm.cuid` field in the data structure identified by the `SharedMemoryID` parameter.

**SHM_SIZE**
Sets the size of the shared memory segment to the value specified by the `shm_segsz` field of the structure specified by the `Buffer` parameter. This value can be larger or smaller than the current size. The limit is the maximum shared-memory segment size. This command is only executed by a process that has an effective user ID equal either to that of a process with the appropriate privileges or to the value of the `shm_perm.uid` or `shm_perm.cuid` field in the data structure identified by the `SharedMemoryID` parameter. This command is not supported for regions created with the environment variable `EXTSHM=ON`. This results in a return value of -1 with `errno` set to `EINVAL`. Attempting to use the `SHM_SIZE` on a shared memory region larger than 256MB or attempting to increase the size of a shared memory region larger than 256MB results in a return value of -1 with `errno` set to `EINVAL`.

**Return Values**
When completed successfully, the `shmctl` subroutine returns a value of 0. Otherwise, it returns a value of -1 and the `errno` global variable is set to indicate the error.

**Error Codes**
The `shmctl` subroutine is unsuccessful if one or more of the following are true:

- **EACCES** The `Command` parameter is equal to the `IPC_STAT` value and read permission is denied to the calling process.
- **EFAULT** The `Buffer` parameter points to a location outside the allocated address space of the process.
- **EINVAL** The `SharedMemoryID` parameter is not a valid shared memory identifier.
- **EINVAL** The `Command` parameter is not a valid command.
- **EINVAL** The `Command` parameter is equal to the `SHM_SIZE` value and the value of the `shm_segsz` field of the structure specified by the `Buffer` parameter is not valid.
- **EINVAL** The `Command` parameter is equal to the `SHM_SIZE` value and the shared memory region was created with the environment variable `EXTSHM=ON`.

Chapter 1. Base Operating System (BOS) Runtime Services (Q-Z) 199
ENOMEM  
The Command parameter is equal to the SHM_SIZE value, and the attempt to change the segment size is unsuccessful because the system does not have enough memory.

EOVERFLOW  
The Command parameter is IPC_STAT and the size of the shared memory region is greater than or equal to 4G bytes. This only happens with 32-bit programs.

EPERM  
The Command parameter is equal to the IPC_RMID or SHM_SIZE value, and the effective user ID of the calling process is not equal to the value of the shm_perm.uid or shm_perm.cuid field in the data structure identified by the SharedMemoryID parameter. The effective user ID of the calling process is not the root user ID.

Related Information
The disclaim subroutine, shmat subroutine, shmdt subroutine, shmget subroutine.

The ipcs command and ipcrm command.

List of Memory Manipulation Services, Subroutines Overview Understanding Memory Mapping in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

shmdt Subroutine

Purpose
Detaches a shared memory segment.

Library
Standard C Library (libc.a)

Syntax
#include <sys/shm.h>

int shmdt (SharedMemoryAddress)
const void * SharedMemoryAddress;

Description
The shmdt subroutine detaches from the data segment of the calling process the shared memory segment located at the address specified by the SharedMemoryAddress parameter.

Mapped file segments are automatically detached when the mapped file is closed. However, you can use the shmdt subroutine to explicitly release the segment register used to map a file. Shared memory segments must be explicitly detached with the shmdt subroutine.

If the file was mapped for writing, the shmdt subroutine updates the mtime and ctime time stamps.

The following limits apply to shared memory:
• Maximum shared-memory segment size is:
  – 256M bytes before AIX 4.3.1
  – 2G bytes for AIX 4.3.1 through AIX 5.1
  – 64G bytes for 64-bit applications for AIX 5.1 and later
• Minimum shared-memory segment size is 1 byte.
• Maximum number of shared memory IDs is 4096 for operating system releases before AIX 4.3.2 and 131072 for AIX 4.3.2 and following.
Parameters

*SharedMemoryAddress*  Specifies the data segment start address of a shared memory segment.

Return Values

When successful, the *shmdt* subroutine returns a value of 0. Otherwise, the shared memory segment at the address specified by the *SharedMemoryAddress* parameter is not detached, a value of 1 is returned, and the *errno* global variable is set to indicate the error.

Error Codes

The *shmdt* subroutine is unsuccessful if the following condition is true:

EINVAL  The value of the *SharedMemoryAddress* parameter is not the data-segment start address of a shared memory segment.

Related Information


The *ipcs* command and *ipcrm* command.

List of Memory Manipulation Services, Subroutines Overview, Understanding Memory Mapping in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

shmget Subroutine

Purpose

Gets shared memory segments.

Library

Standard C Library (*libc.a*)

Syntax

```c
#include <sys/shm.h>

int shmget (Key_t Key, size_t Size, int SharedMemoryFlag);
```

Description

The *shmget* subroutine returns the shared memory identifier associated with the specified *Key* parameter.

The following limits apply to shared memory:

- Maximum shared-memory segment size is:
  - 256M bytes before AIX 4.3.1
  - 2G bytes for AIX 4.3.1 through AIX 5.1
  - 64G bytes for 64-bit applications for AIX 5.1 and later
- Minimum shared-memory segment size is 1 byte.
Maximum number of shared memory IDs is 4096 for operating system releases before AIX 4.3.2, 131072 for releases AIX 4.3.2 through AIX 5.2, and 1048576 for release AIX 5.3 and later.

Parameters

**Key** Specifies either the IPC_PRIVATE value or an IPC key constructed by the ftok subroutine (or by a similar algorithm).

**Size** Specifies the number of bytes of shared memory required.

**SharedMemoryFlag** Constructed by logically ORing one or more of the following values:

- **IPC_CREAT** Creates the data structure if it does not already exist.
- **IPC_EXCL** Causes the shmget subroutine to be unsuccessful if the IPC_CREAT flag is also set, and the data structure already exists.
- **SHM_LGPAGE** Attempts to create the region so it can be mapped through hardware-supported, large-page mechanisms, if enabled. This is purely advisory. For the system to consider this flag, it must be used in conjunction with the SHM_PIN flag and enabled with the vmtune command (\(-L\) to reserve memory for the region (which requires a reboot)) and \(-S\) to enable SHM_PIN). To successfully get large-pages, the user requesting large-page shared memory must have CAP_BYPASS_RAC_VMM capability. This has no effect on shared memory regions created with the EXTSHM=ON environment variable.
- **SHM_PIN** Attempts to pin the shared memory region if enabled. This is purely advisory. For the system to consider this flag, the system must be enable with vmtune command. This has no effect on shared memory regions created with EXTSHM=ON environment variable.
- **S_IRUSR** Permits the process that owns the data structure to read it.
- **S_IWUSR** Permits the process that owns the data structure to modify it.
- **S_IRGRP** Permits the group associated with the data structure to read it.
- **S_IWGRP** Permits the group associated with the data structure to modify it.
- **S_IROTH** Permits others to read the data structure.
- **S_IWOTH** Permits others to modify the data structure.

Values that begin with the S_I prefix are defined in the sys/mode.h file and are a subset of the access permissions that apply to files.

A shared memory identifier, its associated data structure, and a shared memory segment equal in number of bytes to the value of the Size parameter are created for the Key parameter if one of the following is true:

- The Key parameter is equal to the IPC_PRIVATE value.
- The Key parameter does not already have a shared memory identifier associated with it, and the IPC_CREAT flag is set in the SharedMemoryFlag parameter.

Upon creation, the data structure associated with the new shared memory identifier is initialized as follows:
• The shm_perm.cuid and shm_perm.uid fields are set to the effective user ID of the calling process.
• The shm_perm.cgid and shm_perm.gid fields are set to the effective group ID of the calling process.
• The low-order 9 bits of the shm_perm.mode field are set to the low-order 9 bits of the SharedMemoryFlag parameter.
• The shm_segsz field is set to the value of the Size parameter.
• The shm_lpid, shm_nattch, shm_atime, and shm_dtime fields are set to 0.
• The shm_ctime field is set to the current time.

Note: Once created, a shared memory segment is deleted only when the system reboots or by issuing the ipcrm command or using the following shmctl subroutine:

if (shmctl (id, IPC_RMID, 0) == -1)
  perror ("error in closing segment"), exit (1);

Return Values
Upon successful completion, a shared memory identifier is returned. Otherwise, the shmget subroutine returns a value of -1 and sets the errno global variable to indicate the error.

Error Codes
The shmget subroutine is unsuccessful if one or more of the following are true:

EACCES A shared memory identifier exists for the Key parameter, but operation permission as specified by the low-order 9 bits of the SharedMemoryFlag parameter is not granted.
EEXIST A shared memory identifier exists for the Key parameter, and both the IPC_CREAT and IPC_EXCL flags are set in the SharedMemoryFlag parameter.
EINVAL A shared memory identifier does not exist and the Size parameter is less than the system-imposed minimum or greater than the system-imposed maximum.
EINVAL A shared memory identifier exists for the Key parameter, but the size of the segment associated with it is less than the Size parameter, and the Size parameter is not equal to 0.
ENOENT A shared memory identifier does not exist for the Key parameter, and the IPC_CREAT flag is not set in the SharedMemoryFlag parameter.
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 meet the request.
ENOSPC A shared memory identifier will be created, but the system-imposed maximum of shared memory identifiers allowed will be exceeded.

Related Information
The ftok subroutine, mmap subroutine, munmap subroutine, shmat subroutine, shmctl subroutine, shmdt subroutine.
The ipcs command and ipcrm command.

List of Memory Manipulation Services | Subroutines Overview | Understanding Memory Mapping in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

sigaction, sigvec, or signal Subroutine

Purpose
Specifies the action to take upon delivery of a signal.

Libraries

sigaction Standard C Library (libc.a)
Signal, sigvec

Standard C Library (libc.a);

Berkeley Compatibility Library (libbsd.a)

Syntax

#include <signal.h>

int sigaction ( Signal, Action, OAction)
int Signal;
struct sigaction *Action, *OAction;

int sigvec (Signal, Invec, Outvec)
int Signal;
struct sigvec *Invec, *Outvec;

void (*signal (Signal, Action)) ()
int Signal;
void (*Action) (int);

Description

The sigaction subroutine allows a calling process to examine and change the action to be taken when a specific signal is delivered to the process issuing this subroutine.

In multi-threaded applications using the threads library (libpthreads.a), signal actions are common to all threads within the process. Any thread calling the sigaction subroutine changes the action to be taken when a specific signal is delivered to the threads process, that is, to any thread within the process.

Note: The sigaction subroutine must not be used concurrently to the sigwait subroutine on the same signal.

The Signal parameter specifies the signal. If the Action parameter is not null, it points to a sigaction structure that describes the action to be taken on receipt of the Signal parameter signal. If the OAction parameter is not null, it points to a sigaction structure in which the signal action data in effect at the time of the sigaction subroutine call is returned. If the Action parameter is null, signal handling is unchanged; thus, the call can be used to inquire about the current handling of a given signal.

The sigaction structure has the following fields:

<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>void(*) (int, siginfo_t *, void *)</td>
<td>sa_sigaction</td>
<td>Signal-catching function.</td>
</tr>
</tbody>
</table>

The sa_handler field can have a SIG_DFL or SIG_IGN value, or it can be a pointer to a function. A SIG_DFL value requests default action to be taken when a signal is delivered. A value of SIG_IGN
requests that the signal have no effect on the receiving process. A pointer to a function requests that the signal be caught; that is, the signal should cause the function to be called. These actions are more fully described in “Parameters”.

When a signal is delivered to a thread, if the action of that signal specifies termination, stop, or continue, the entire process is terminated, stopped, or continued, respectively.

If the SA_SIGINFO flag (see below) is cleared in the sa_flags field of the sigaction structure, the sa_handler field identifies the action to be associated with the specified signal. If the SA_SIGINFO flag is set in the sa_flags field, the sa_sigaction field specifies a signal-catching function. If the SA_SIGINFO bit is cleared and the sa_handler field specifies a signal-catching function, or if the SA_SIGINFO bit is set, the sa_mask field identifies a set of signals that will be added to the signal mask of the thread before the signal-catching function is invoked.

The sa_mask field can be used to specify that individual signals, in addition to those in the process signal mask, be blocked from being delivered while the signal handler function specified in the sa_handler field is operating. The sa_flags field can have the SA_ONSTACK, SA_OLDSTYLE, or SA_NOCLDSTOP bits set to specify further control over the actions taken on delivery of a signal.

If the SA_ONSTACK bit is set, the system runs the signal-catching function on the signal stack specified by the sigstack subroutine. If this bit is not set, the function runs on the stack of the process to which the signal is delivered.

If the SA_OLDSTYLE bit is set, the signal action is set to SIG_DFL label prior to calling the signal-catching function. This is supported for compatibility with old applications, and is not recommended since the same signal can recur before the signal-catching subroutine is able to reset the signal action and the default action (normally termination) is taken in that case.

If a signal for which a signal-catching function exists is sent to a process while that process is executing certain subroutines, the call can be restarted if the SA_RESTART bit is set for each signal. The only affected subroutines are the following:

- read, readx, readv, or readvx
- write, writex, writev, or writevx
- ioctl or ioctlx
- fcntl or flock
- wait, wait3, or waitpid

Other subroutines do not restart and return EINTR label, independent of the setting of the SA_RESTART bit.

If SA_SIGINFO is cleared and the signal is caught, the signal-catching function will be entered as:

```c
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:

```c
void func(int signo, siginfo_t *info, void *context);
```

where two additional arguments are passed to the signal catching function.

The second argument 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 nonzero, 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, 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 `signal.h` header description contains information about the signal specific contents of the elements of the `siginfo_t` type. If `SA_NOCLDWAIT` is 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. When SIGCHLD is set to SIG_IGN, the signal is ignored and any zombie children of the process will be cleaned up.

If `SA_RESETHAND` is 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.

If `SA_NODEFER` is 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, the `SA_NOCLDSTOP` flag is not set in `sa_flags`, and the implementation supports the SIGCHLD signal, 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`, 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 including the signal being delivered. If 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 `act` are unspecified, and in particular `act->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` through 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.

If `SA_SIGINFO` is not set in `sa_flags`, then the disposition of subsequent occurrences of `sig` when it is already pending is implementation-dependent; the signal-catching function will be invoked with a single argument.

The `sigvec` and `signal` subroutines are provided for compatibility to older operating systems. Their function is a subset of that available with `sigaction`. 

206  Technical Reference, Volume 2: Base Operating System and Extensions
The **sigvec** subroutine uses the **sigvec** structure instead of the **sigaction** structure. The **sigvec** structure specifies a mask as an `int` instead of a `sigset_t`. The mask for the **sigvec** subroutine is constructed by setting the `i`-th bit in the mask if signal `i` is to be blocked. Therefore, the **sigvec** subroutine only allows signals between the values of 1 and 31 to be blocked when a signal-handling function is called. The other signals are not blocked by the signal-handler mask.

The **sigvec** structure has the following members:
```
int (*sv_handler)();
/* signal_handler */
int sv_mask;
/* signal mask */
int sv_flags;
/* flags */
```

The **sigvec** subroutine in the `libbsd.a` library interprets the `SV_INTERRUPT` flag and inverts it to the `SA_RESTART` flag of the **sigaction** subroutine. The **sigvec** subroutine in the `libc.a` library always sets the `SV_INTERRUPT` flag regardless of what was passed in the **sigvec** structure.

The **sigvec** subroutine in the `libbsd.a` library interprets the `SV_INTERRUPT` flag and inverts it to the `SA_RESTART` flag of the **sigaction** subroutine. The **sigvec** subroutine in the `libc.a` library always sets the `SV_INTERRUPT` flag regardless of what was passed in the **sigvec** structure.

The **signal** subroutine in the `libc.a` library allows an action to be associated with a signal. The `Action` parameter can have the same values that are described for the `sv_handler` field in the **sigaction** structure of the **sigaction** subroutine. However, no signal handler mask or flags can be specified; the **signal** subroutine implicitly sets the signal handler mask to additional signals and the flags to be `SA_OLDSTYLE`.

Upon successful completion of a **signal** call, the value of the previous signal action is returned. If the call fails, a value of -1 is returned and the `errno` global variable is set to indicate the error as in the **sigaction** call.

The **signal** in `libc.a` does not set the `SA_RESTART` flag. It sets the signal mask to the signal whose action is being specified, and sets flags to `SA_OLDSTYLE`. The Berkeley Software Distribution (BSD) version of **signal** sets the `SA_RESTART` flag and preserves the current settings of the signal mask and flags. The BSD version can be used by compiling with the Berkeley Compatibility Library (`libbsd.a`).

The **signal** in `libc.a` does not set the `SA_RESTART` flag. It sets the signal mask to the signal whose action is being specified, and sets flags to `SA_OLDSTYLE`. The Berkeley Software Distribution (BSD) version of **signal** sets the `SA_RESTART` flag and preserves the current settings of the signal mask and flags. The BSD version can be used by compiling with the Berkeley Compatibility Library (`libbsd.a`).

### Parameters

**Signal** Defines the signal. The following list describes signal names and the specification for each. The value of the **Signal** parameter can be any signal name from this list or its corresponding number except the **SIGKILL** name. If you use the signal name, you must include the **signal.h** file, because the name is correlated in the file with its corresponding number.

**Note:** The symbols in the following list of signals represent these actions:

- `*` Specifies the default action that includes creating a core dump file.
- `@` Specifies the default action that stops the process receiving these signals.
- `!` Specifies the default action that restarts or continues the process receiving these signals.
- `+` Specifies the default action that ignores these signals.
- `%` Indicates a likely shortage of paging space.
See *Terminal Programming* for more information on the use of these signals.

**SIGHUP**
Hang-up. (1)

**SIGINT**
Interrupt. (2)

**SIGQUIT**
Quit. (3*)

**SIGILL**
Invalid instruction (not reset when caught). (4*)

**SIGTRAP**
Trace trap (not reset when caught). (5*)

**SIGIOT**
End process (see the *abort* subroutine). (6*)

**SIGEMT**
EMT instruction. (7*)

**SIGFPE**
Arithmetic exception, integer divide by 0, or floating-point exception. (8*)

**SIGKILL**
Kill (cannot be caught or ignored). (9)

**SIGBUS**
Specification exception. (10*)

**SIGSEGV**
Segmentation violation. (11*)

**SIGSYS**
Parameter not valid to subroutine. (12*)

**SIGPIPE**
Write on a pipe when there is no process to read it. (13)

**SIGALRM**
Alarm clock. (14)

**SIGTERM**
Software termination signal. (15)

**SIGURG**
Urgent condition on I/O channel. (16+)

**SIGSTOP**
Stop (cannot be caught or ignored). (17@)

**SIGTSTP**
Interactive stop. (18@)

**SIGCONT**
Continue if stopped. (19!)

**SIGCHLD**
To parent on child stop or exit. (20+)

**SIGTIN**
Background read attempted from control terminal. (21@)

**SIGTTOU**
Background write attempted from control terminal. (22@)
SIGIO   Input/output possible or completed. (23+)
SIGXCPU CPU time limit exceeded (see the setrlimit subroutine). (24)
SIGXFSZ File size limit exceeded (see the setrlimit subroutine). (25)
reserved (26)
SIGMSG Input data has been stored into the input ring buffer. (27#)
SIGWINCH Window size change. (28+)
SIGPWR Power-fail restart. (29+)
SIGUSR1 User-defined signal 1. (30)
SIGUSR2 User-defined signal 2. (31)
SIGPROF Profiling timer expired. (see the setitimer subroutine). (32)
SIGDANGER Paging space low. (33+%)
SIGVTALRM Virtual time alarm (see the setitimer subroutine). (34)
SIGMIGRATE Migrate process. (35)
SIGPRE Programming exception (user defined). (36)
reserved (37-58)
SIGGRANT Monitor access wanted. (60#)
SIGRETRACT Monitor access should be relinquished. (61#)
SIGSOUND A sound control has completed execution. (62#)
SIGSAK Secure attention key. (63)

**Action** Points to a sigaction structure that describes the action to be taken upon receipt of the Signal/parameter signal.

The three types of actions that can be associated with a signal (SIG_DFL, SIG_IGN, or a pointer to a function) are described as follows:

- **SIG_DFL** Default action: signal-specific default action.
Except for those signal numbers marked with a plus sign, at sign, or exclamation point, the default action for a signal ends the receiving process with all of the consequences described in the _exit subroutine. In addition, a memory image file is created in the current directory of the receiving process if an asterisk appears with a Signal parameter and the following conditions are met:

- The saved user ID and the real user ID of the receiving process are equal.
- An ordinary file named core exists in the current directory and is writable, or it can be created. If the file is created, it must have the following properties:
  - The access permission code 0666 (0x1B6), modified by the file-creation mask (see the umask subroutine)
  - A file owner ID that is the same as the effective user ID of the receiving process
  - A file group ID that is the same as the effective group ID of the receiving process.

For signal numbers marked with an exclamation point, the default action restarts the receiving process if it has stopped, or continues to run the receiving process.

For signal numbers marked with an at sign, the default action stops the execution of the receiving process temporarily. When a process stops, a SIGCHLD signal is sent to its parent process, unless the parent process has set the SA_NOCLDSTOP bit. While a process has stopped, any additional signals that are sent are not delivered until the process has started again. An exception to this is the SIGKILL signal, which always terminates the receiving process. Another exception is the SIGCONT signal, which always causes the receiving process to restart or continue running. A process whose parent process has ended is sent a SIGKILL signal if the SIGTSTP, SIGTTIN, or SIGTTOU signals are generated for that process.

For signal numbers marked with a plus sign, the default action ignores the signal. In this case, the delivery of a signal does not affect the receiving process.

If a signal action is set to SIG_DFL while the signal is pending, the signal remains pending.

- SIG_IGN Ignore signal.
  Delivery of the signal does not affect the receiving process. If a signal action is set to the SIG_IGN action while the signal is pending, the pending signal is discarded.

An exception to this is the SIGCHLD signal whose SIG_DFL action ignores the signal. 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.

Note: The SIGKILL and SIGSTOP signals cannot be ignored.

- Pointer to a function, catch signal.
  Upon delivery of the signal, the receiving process runs the signal-catch function specified by the pointer to function. The signal-handler subroutine can be declared as follows:

```c
handler(signal, code, scp)
int signal, code;
struct sigcontext *scp;
```

The Signal parameter is the signal number. The Code parameter is provided only for compatibility with other UNIX-compatible systems. The Code parameter value is always 0. The SCP parameter points to the sigcontext structure that is later used to restore the previous execution context of the process. The sigcontext structure is defined in the signal.h file.

A new signal mask is calculated and installed for the duration of the signal-catch function (or until sigprocmask or sigsuspend subroutine is made). This mask is formed by joining the process-signal mask (the mask associated with the action for the signal being delivered) and the mask corresponding to the signal being delivered. The mask associated with the signal-catch function is not allowed to block those signals that cannot be ignored. This is
enforced by the kernel without causing an error to be indicated. If and when the signal-catching function returns, the original signal mask is restored (modified by any \texttt{sigprocmask} calls that were made since the signal-catching function was called) and the receiving process resumes execution at the point it was interrupted.

The signal-catching function can cause the process to resume in a different context by calling the \texttt{longjmp} subroutine. When the \texttt{longjmp} subroutine is called, the process leaves the signal stack, if it is currently on the stack, and restores the process signal mask to the state when the corresponding \texttt{setjmp} subroutine was made.

Once an action is installed for a specific signal, it remains installed until another action is explicitly requested (by another call to the \texttt{sigaction} subroutine), or until one of the \texttt{exec} subroutines is called. An exception to this is when the \texttt{SA_OLDSTYLE} bit is set. In this case the action of a caught signal gets set to the \texttt{SIG_DFL} action before the signal-catching function for that signal is called.

If a signal action is set to a pointer to a function while the signal is pending, the signal remains pending.

When signal-catching functions are invoked asynchronously with process execution, the behavior of some of the functions defined by this standard is unspecified if they are called from a signal-catching function. The following set of functions are reentrant with respect to signals; that is, applications can invoke them, without restriction, from signal-catching functions:

\begin{verbatim}
_exit
access
alarm
cfgetispeed
cfgetospeed
cfsetispeed
cfsetospeed
chdir
chmod
chown
close
creat
dup
dup2
exec
execl
execve
fcntl
fpathconf
fstat
getegid
geteuid
getgid
\end{verbatim}
statx
sysconf
tcdrain
tcf\low
tcflush
tcgetattr
tcgetpgrp
tcsendbreak
tcsetattr
tcsetpgrp
time	
times	
umask
uname
unlink
ustat
utime
wait
waitpid
write

All other subroutines should not be called from signal-catching functions since their behavior is undefined.

*OA*ction
Points to a *sigaction* structure in which the signal action data in effect at the time of the *sigaction* subroutine is returned.

**Invec** Points to a *sigvec* structure that describes the action to be taken upon receipt of the *Signal* parameter signal.

**Outvec** Points to a *sigvec* structure in which the signal action data in effect at the time of the *sigvec* subroutine is returned.

**Action** Specifies the action associated with a signal.

**Return Values**
Upon successful completion, the *sigaction* subroutine returns a value of 0. Otherwise, a value of *SIG_ERR* is returned and the *errno* global variable is set to indicate the error.

**Error Codes**
The *sigaction* subroutine is unsuccessful and no new signal handler is installed if one of the following occurs:

**EFAULT** The *Action* or **OA**ction parameter points to a location outside of the allocated address space of the process.
EINVAL The Signal parameter is not a valid signal number.
EINVAL An attempt was made to ignore or supply a handler for the SIGKILL, SIGSTOP, and SIGCONT signals.

Related Information
The acct subroutine, exit, exit, or atexit subroutine, getinterval, incinterval, absinterval, resinc, resabs, alarm,ualarm, gettimer, or settimer subroutine, getrlimit, setrlimit, or vlimit subroutine, kill subroutine, longjmp or setjmp subroutine, pause subroutine, ptrace subroutine, sigpause or sigsuspend subroutine, sigprocmask, sigsetmask, or sigblock subroutine, sigstack subroutine, sigwait subroutine, umask subroutine, wait, waitpid, or wait3 subroutine.

The kill command.
The core file.

Signal Management in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs provides more information about signal management in multi-threaded processes.

sigaltstack Subroutine

Purpose
Allows a thread to define and examine the state of an alternate stack for signal handlers.

Library
(libc.a)

Syntax
#include <signal.h>

int sigaltstack(const stack_t *ss, stack_t *oss);

Description
The sigaltstack subroutine allows a thread 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 null pointer, it points to a stack_t structure that specifies the alternate signal stack that will take effect upon return from sigaltstack subroutine. 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.

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 sigaltstack subroutine. The ss_sp and ss_size members specify the address and size of the 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 or it fails. This flag must not be modified by processes.

SS_DISABLE  The alternate signal stack is currently disabled.

The value of 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.

After a successful call to one of the exec functions, there are no alternate stacks in the new process image.

Parameters

ss  A pointer to a stack_t structure specifying the alternate stack to use during signal handling.
oss  A pointer to a stack_t structure that will indicate the alternate stack currently in use.

Return Values

Upon successful completion, sigaltstack subroutine returns 0. Otherwise, it returns -1 and set errno to indicate the error.

-1  Not successful and the errno global variable is set to one of the following error codes.

Error Codes

EINVAL  The ss parameter is not a null pointer, and the ss_flags member pointed to by ss contains flags other that SS_DISABLE.
ENOMEM  The size of the alternate stack area is less than MINSIGSTKSZ.
EPERM  An attempt was made to modify an active stack.

Related Information

The sigaction (“sigaction, sigvec, or signal Subroutine” on page 203) subroutine, sigsetjmp (“sigsetjmp or siglongjmp Subroutine” on page 226) subroutine.

sigemptyset, sigfillset, sigaddset, sigdelset, or sigismember Subroutine

Purpose

Creates and manipulates signal masks.

Library

Standard C Library (libc.a)

Syntax

#include <signal.h>

int sigemptyset (Set)
sigset_t *Set;
The `sigemptyset`, `sigfillset`, `sigaddset`, `sigdelset`, and `sigismember` subroutines manipulate sets of signals. These functions operate on data objects addressable by the application, not on any set of signals known to the system, such as the set blocked from delivery to a process or the set pending for a process.

The `sigemptyset` subroutine initializes the signal set pointed to by the `Set` parameter such that all signals are excluded. The `sigfillset` subroutine initializes the signal set pointed to by the `Set` parameter such that all signals are included. A call to either the `sigfillset` or `sigemptyset` subroutine must be made at least once for each object of the `sigset_t` type prior to any other use of that object.

The `sigaddset` and `sigdelset` subroutines respectively add and delete the individual signal specified by the `SignalNumber` parameter from the signal set specified by the `Set` parameter. The `sigismember` subroutine tests whether the `SignalNumber` parameter is a member of the signal set pointed to by the `Set` parameter.

### Parameters

- **Set**: Specifies the signal set.
- **SignalNumber**: Specifies the individual signal.

### Examples

To generate and use a signal mask that blocks only the `SIGINT` signal from delivery, enter:

```c
#include <signal.h>

int return_value;
int newset;

newset_p = &newset;
return_value = sigprocmask (SIG_SETMASK, newset_p, NULL);
```

### Return Values

Upon successful completion, the `sigismember` subroutine returns a value of 1 if the specified signal is a member of the specified set, or the value of 0 if not. Upon successful completion, the other subroutines return a value of 0. For all the preceding subroutines, if an error is detected, a value of -1 is returned and the `errno` global variable is set to indicate the error.

### Error Codes

The `sigfillset`, `sigdelset`, `sigismember`, and `sigaddset` subroutines are unsuccessful if the following is true:
The value of the SignalNumber parameter is not a valid signal number.

Related Information
The sigaction, sigvec, or signal subroutine, sigprocmask subroutine, sigsuspend subroutine.

siginterrupt Subroutine

Purpose
Sets restart behavior with respect to signals and subroutines.

Library
Standard C Library (libc.a)

Syntax
int siginterrupt (Signal, Flag);

Description
The siginterrupt subroutine is used to change the subroutine restart behavior when a subroutine is interrupted by the specified signal. If the flag is false (0), subroutines are restarted if they are interrupted by the specified signal and no data has been transferred yet.

If the flag is true (1), the restarting of subroutines is disabled. If a subroutine is interrupted by the specified signal and no data has been transferred, the subroutine will return a value of -1 with the errno global variable set to EINTR. Interrupted subroutines that have started transferring data return the amount of data actually transferred. Subroutine interrupt is the signal behavior found on 4.1 BSD and AT&T System V UNIX systems.

Note that the BSD signal-handling semantics are not altered in any other way. Most notably, signal handlers always remain installed until explicitly changed by a subsequent sigaction or sigvec call, and the signal mask operates as documented in the sigaction subroutine. Programs can switch between restartable and interruptible subroutine operations as often as desired in the running of a program.

Issuing a siginterrupt call during the running of a signal handler causes the new action to take place on the next signal caught.

Restart does not occur unless it is explicitly specified with the sigaction or sigvec subroutine in the libc.a library.

This subroutine uses an extension of the sigvec subroutine that is not available in the BSD 4.2; hence, it should not be used if backward compatibility is needed.

Parameters
Signal Indicates the signal.
Flag Indicates true or false.
Return Values
A value of 0 indicates that the call succeeded. A value of -1 indicates that the supplied signal number is not valid.

Related Information
The sigaction or sigvec subroutine, sigpause subroutine, sigsetmask or sigblock subroutine.

signbit Macro

Purpose
Tests the sign.

Syntax
#include <math.h>

int signbit (x)

real-floating x;

Description
The signbit macro determines whether the sign of its argument value is negative. NaNs, zeros, and infinities have a sign bit.

Parameters
x Specifies the value to be tested.

Return Values
The signbit macro returns a nonzero value if the sign of its argument value is negative.

Related Information
class, _class, finite, isnan, or unordered Subroutines, fpclassify Subroutine, isnormal Subroutine, ldexp Subroutine in AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 1.

sigpending Subroutine

Purpose
Returns a set of signals that are blocked from delivery.

Library
Standard C Library (libc.a)

Syntax
#include <signal.h>
Description
The **sigpending** subroutine stores a set of signals that are blocked from delivery and pending for the calling thread, in the space pointed to by the *Set* parameter.

Parameters

*Set* Specifies the set of signals.

Return Values

Upon successful completion, the **sigpending** subroutine returns a value of 0. Otherwise, a value of -1 is returned and the **errno** global variable is set to indicate the error.

Error Codes

The **sigpending** subroutine is unsuccessful if the following is true:

**EINVAL** The input parameter is outside the user's address space.

Related Information

The **sigprocmask** (**`sigprocmask, sigsetmask, or sigblock Subroutine`**) subroutine.

**sigprocmask, sigsetmask, or sigblock Subroutine**

Purpose

Sets the current signal mask.

Library

Standard C Library (**libc.a**)

Syntax

```c
#include <signal.h>

int sigprocmask (How, Set, OSet)
int How;
const sigset_t *Set;
sigset *OSet;

int sigsetmask (SignalMask)
int SignalMask;

int sigblock (SignalMask)
int SignalMask;
```

Description

**Note:** The **sigprocmask**, **sigsetmask**, and **sigblock** subroutines must not be used in a multi-threaded application. The **sigthreadmask** (**`sigthreadmask Subroutine`** on page 229) subroutine must be used instead.
The `sigprocmask` subroutine is used to examine or change the signal mask of the calling thread.

The subroutine is used to examine or change the signal mask of the calling process.

Typically, you should use the `sigprocmask(SIG_BLOCK)` subroutine to block signals during a critical section of code. Then use the `sigprocmask(SIG_SETMASK)` subroutine to restore the mask to the previous value returned by the `sigprocmask(SIG_BLOCK)` subroutine.

If there are any pending unblocked signals after the call to the `sigprocmask` subroutine, at least one of those signals will be delivered before the `sigprocmask` subroutine returns.

The `sigprocmask` subroutine does not allow the SIGKILL or SIGSTOP signal to be blocked. If a program attempts to block either signal, the `sigprocmask` subroutine gives no indication of the error.

**Parameters**

`How` Indicates the manner in which the set is changed. It can have one of the following values:

- `SIG_BLOCK` The resulting set is the union of the current set and the signal set pointed to by the `Set` parameter.
- `SIG_UNBLOCK` The resulting set is the intersection of the current set and the complement of the signal set pointed to by the `Set` parameter.
- `SIG_SETMASK` The resulting set is the signal set pointed to by the `Set` parameter.

`Set` Specifies the signal set. If the value of the `Set` parameter is not null, it points to a set of signals to be used to change the currently blocked set. If the value of the `Set` parameter is null, the value of the `How` parameter is not significant and the process signal mask is unchanged. Thus, the call can be used to inquire about currently blocked signals.

`OSet` If the `OSet` parameter is not the null value, the signal mask in effect at the time of the call is stored in the space pointed to by the `OSet` parameter.

`SignalMask` Specifies the signal mask of the process.

**Compatibility Interfaces**

The `sigsetmask` subroutine allows changing the process signal mask for signal values 1 to 31. This same function can be accomplished for all values with the `sigprocmask(SIG_SETMASK)` subroutine. The signal of value \( i \) will be blocked if the \( i \)-th bit of `SignalMask` parameter is set.

Upon successful completion, the `sigsetmask` subroutine returns the value of the previous signal mask. If the subroutine fails, a value of -1 is returned and the `errno` global variable is set to indicate the error as in the `sigprocmask` subroutine.

The `sigblock` subroutine allows signals with values 1 to 31 to be logically ORed into the current process signal mask. This same function can be accomplished for all values with the `sigprocmask(SIG_BLOCK)` subroutine. The signal of value \( i \) will be blocked, in addition to those currently blocked, if the \( i \)-th bit of the `SignalMask` parameter is set.

It is not possible to block a SIGKILL or SIGSTOP signal using the `sigblock` or `sigsetmask` subroutine. This restriction is silently imposed by the system without causing an error to be indicated.

Upon successful completion, the `sigblock` subroutine returns the value of the previous signal mask. If the subroutine fails, a value of -1 is returned and the `errno` global variable is set to indicate the error as in the `sigprocmask` subroutine.
Return Values
Upon completion, a value of 0 is returned. If the sigprocmask subroutine fails, the signal mask of the process is unchanged, a value of -1 is returned, and the global variable errno is set to indicate the error.

Error Codes
The sigprocmask subroutine is unsuccessful if the following is true:

EPERM The user does not have the privilege to change the signal's mask.
EINVAL The value of the How parameter is not equal to one of the defined values.
EFAULT The user's mask is not in the process address space.

Examples
To set the signal mask to block only the SIGINT signal from delivery, enter:

```
#include <signal.h>

int return_value;
sigset_t newset;
sigset_t *newset_p;
...
newset_p = &newset;
sigemptyset(newset_p);
sigaddset(newset_p, SIGINT);
return_value = sigprocmask(SIG_SETMASK, newset_p, NULL);
```

Related Information
The kill or killpg subroutine, sigaction, sigvec, or signal subroutine, sigaddset, sigdelset, sigemptyset, sigfillset, sigismember subroutine, sigpause subroutine, sigpending subroutine, sigqueue subroutine.

sigqueue Subroutine

Purpose
Queues a signal to a process.

Library
Standard C Library (libc.a)

Syntax
```
#include <signal.h>

int sigqueue(pid_t pid, int signo, const union sigval value);
```

**Description**

The `sigqueue` subroutine causes the signal specified by the `signo` parameter to be sent with the value specified by the `value` parameter to the process specified by the `pid` parameter. If the `signo` parameter is zero, error checking is performed but no signal is actually sent. This can be used to check the validity of the `pid` parameter.

The conditions required for a process to have permission to queue a signal to another process are the same as for the `kill` subroutine.

The `sigqueue` subroutine returns immediately. If `SA_SIGINFO` is set by the receiving process for the specified signal, and if the resources are available to queue the signal, the signal is queued and sent to the receiving process. If `SA_SIGINFO` is not set for the `signo` parameter, the signal is sent at least once to the receiving process.

If multiple signals in the range `SIGRTMIN` to `SIGRTMAX` should be available for delivery, the lowest numbered of them will be delivered first.

**Parameters**

- `pid` Specifies the process to which a signal is to be sent.
- `signo` Specifies the signal number.
- `value` Specifies the value to be sent with the signal.

**Return Values**

Upon successful completion the `sigqueue` subroutine returns a zero. If unsuccessful, it returns a -1 and sets the `errno` variable to indicate the error.

**Error Code**

The `sigqueue` subroutine will fail if:

- **EAGAIN** No resources are available to queue the signal. The process has already queued `SIGQUEUE_MAX` signals that are still pending at the receiver(s), or a system-wide resource limit has been exceeded.
- **EINVAL** The value of the `signo` parameter is an invalid or unsupported signal number, or if the selected signal can either stop or continue the receiving process. AIX does not support queuing of the following signals: SIGKILL, SIGSTOP, SIGTSTOP, SIGCONT, SIGTTIN, SIGTTOU, and SIGCLD.
- **EPERM** The process does not have the appropriate privilege to send the signal to the receiving process.
- **ESRCH** The process specified by the `pid` parameter does not exist.

**Related Information**

- "sigtimedwait and sigwaitinfo Subroutine" on page 231
- "sigaction, sigvec, or signal Subroutine" on page 203

**sigset, sighold, sigrelse, or sigignore Subroutine**

**Purpose**

Enhance the signal facility and provide signal management.

**Library**

Standard C Library (libc.a)
Syntax

```c
#include <signal.h>
void (*sigset(Signal, Function))();
int Signal;
void (*Function)();
int sighold(Signal);
int Signal;
int sigrelse(Signal);
int Signal;
int sigignore(Signal);
int Signal;
```

Description

The `sigset`, `sighold`, `sigrelse`, and `sigignore` subroutines enhance the signal facility and provide signal management for application processes.

The `sigset` subroutine specifies the system signal action to be taken upon receiving a `Signal` parameter.

The `sighold` and `sigrelse` subroutines establish critical regions of code. A call to the `sighold` subroutine is analogous to raising the priority level and deferring or holding a signal until the priority is lowered by `sigrelse`. The `sigrelse` subroutine restores the system signal action to the action that was previously specified by the `sigset` structure.

The `sigignore` subroutine sets the action for the `Signal` parameter to `SIG_IGN`.

The other signal management routine, `signal`, should not be used in conjunction with these routines for a particular signal type.
Parameters

Signal  Specifies the signal. The Signal parameter can be assigned any one of the following signals:

SIGHUP    Hang up
SIGINT    Interrupt
SIGQUIT   Quit*
SIGILL    Illegal instruction (not reset when caught)*
SIGTRAP   Trace trap (not reset when caught)*
SIGABRT   Abort*
SIGFPE    Floating point exception*, or arithmetic exception, integer divide by 0
SIGSYS    Bad argument to routine*
SIGPIPE   Write on a pipe with no one to read it
SIGALRM   Alarm clock
SIGTERM   Software termination signal
SIGUSR1   User-defined signal 1
SIGUSR2   User-defined signal 2.

* The default action for these signals is an abnormal termination.

For portability, application programs should use or catch only the signals listed above. Other signals are hardware-dependant and implementation-dependant and may have very different meanings or results across systems. For example, the System V signals (SIGEMT, SIGBUS, SIGSEGV, and SIGIOT) are implementation-dependent and are not listed above. Specific implementations may have other implementation-dependent signals.
Function

Specifies the choice. The Function parameter is declared as a type pointer to a function returning void. The Function parameter is assigned one of four values: SIG_DFL, SIG_IGN, SIG_HOLD, or an address of a signal-catching function. Definitions of the actions taken by each of the values are:

SIG_DFL

Terminate process upon receipt of a signal.

Upon receipt of the signal specified by the Signal parameter, the receiving process is to be terminated with all of the consequences outlined in the exit subroutine. In addition, if Signal is one of the signals marked with an asterisk above, implementation-dependent abnormal process termination routines, such as a core dump, can be invoked.

SIG_IGN

Ignore signal.

Any pending signal specified by the Signal parameter is discarded. A pending signal is a signal that has occurred but for which no action has been taken. The system signal action is set to ignore future occurrences of this signal type.

SIG_HOLD

Hold signal.

The signal specified by the Signal parameter is to be held. Any pending signal of this type remains held. Only one signal of each type is held.

address

Catch signal.

Upon receipt of the signal specified by the Signal parameter, the receiving process is to execute the signal-catching function pointed to by the Function parameter. Any pending signal of this type is released. This address is retained across calls to the other signal management functions, sighold and sigrelse. The signal number Signal is passed as the only argument to the signal-catching function. Before entering the signal-catching function, the value of the Function parameter for the caught signal is set to SIG_HOLD. During normal return from the signal-catching handler, the system signal action is restored to the Function parameter and any held signal of this type is released. If a nonlocal goto (see the setjmp subroutine) is taken, the sigrelse subroutine must be invoked to restore the system signal action and to release any held signal of this type.

Upon return from the signal-catching function, the receiving process will resume execution at the point at which it was interrupted, except for implementation-defined signals in which this may not be true.

When a signal to be caught occurs during a nonatomic operation such as a call to the read, write, open, or ioctl subroutine on a slow device (such as a terminal); during a pause subroutine; during a wait subroutine that does not return immediately, the signal-catching function is executed. The interrupted routine then returns a value of -1 to the calling process with the errno global variable set to EINTR.

Return Values

Upon successful completion, the sigset subroutine returns the previous value of the system signal action for the specified Signal. Otherwise, it returns SIG_ERR and the errno global variable is set to indicate the error.

For the sighold, sigrelse, and sigignore subroutines, a value of 0 is returned upon success. Otherwise, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes

The sigset, sighold, sigrelse, or sigignore subroutine is unsuccessful if the following is true:

EINVAL    The Signal value is either an illegal signal number, or the default handling of Signal cannot be changed.
Related Information

The \texttt{exit} subroutine, \texttt{kill} subroutine, \texttt{setjmp} subroutine, \texttt{signal} subroutine, \texttt{wait} subroutine, \texttt{wait3} subroutine, \texttt{wait364} subroutine.

\section*{sigsetjmp or siglongjmp Subroutine}

\subsection*{Purpose}
Saves or restores stack context and signal mask.

\subsection*{Library}
Standard C Library (\texttt{libc.a})

\subsection*{Syntax}

\begin{verbatim}
#include <setjmp.h>

int sigsetjmp (Environment, SaveMask)
  sigjmp_buf Environment;
  int SaveMask;

void siglongjmp (Environment, Value)
  sigjmp_buf Environment;
  int Value;
\end{verbatim}

\subsection*{Description}
The \texttt{sigsetjmp} subroutine saves the current stack context, and if the value of the \texttt{SaveMask} parameter is not 0, the \texttt{sigsetjmp} subroutine also saves the current signal mask of the process as part of the calling environment.

The \texttt{siglongjmp} subroutine restores the saved signal mask only if the \texttt{Environment} parameter was initialized by a call to the \texttt{sigsetjmp} subroutine with a nonzero \texttt{SaveMask} parameter argument.

\subsection*{Parameters}

\begin{itemize}
  \item \texttt{Environment} Specifies an address for a \texttt{sigjmp_buf} structure.
  \item \texttt{SaveMask} Specifies the flag used to determine if the signal mask is to be saved.
  \item \texttt{Value} Specifies the return value from the \texttt{siglongjmp} subroutine.
\end{itemize}

\subsection*{Return Values}
The \texttt{sigsetjmp} subroutine returns a value of 0. The \texttt{siglongjmp} subroutine returns a nonzero value.

\subsection*{Related Information}
The \texttt{setjmp} subroutine, \texttt{sigaction} subroutine, \texttt{sigprocmask} subroutine, \texttt{sigsuspend} subroutine.
sigstack Subroutine

Purpose
Sets and gets signal stack context.

Library
Standard C Library (libc.a)

Syntax
#include <signal.h>

int sigstack (InStack, OutStack);
struct sigstack *InStack, *OutStack;

Description
The sigstack subroutine defines an alternate stack on which signals are to be processed.

When a signal occurs and its handler is to run on the signal stack, the system checks to see if the process
is already running on that stack. If so, it continues to do so even after the handler returns. If not, the signal
handler runs on the signal stack, and the original stack is restored when the handler returns.

Use the sigvec or sigaction subroutine to specify whether a given signal-handler routine is to run on the
signal stack.

Attention: A signal stack does not automatically increase in size as a normal stack does. If the stack
overflows, unpredictable results can occur.

Parameters

InStack Specifies the stack pointer of the new signal stack.

If the value of the InStack parameter is nonzero, it points to a sigstack structure, which has the
following members:
caddr_t ss_sp;
int ss_onstack;

The value of InStack->ss_sp specifies the stack pointer of the new signal stack. Since stacks grow
from numerically greater addresses to lower ones, the stack pointer passed to the sigstack
subroutine should point to the numerically high end of the stack area to be used.
InStack->ss_onstack should be set to a value of 1 if the process is currently running on that stack;
otherwise, it should be a value of 0.

OutStack Points to structure where current signal stack state is stored.

If the value of the OutStack parameter is nonzero, it points to a sigstack structure into which the
sigstack subroutine stores the current signal stack state.

If the value of the OutStack parameter is 0, the previous signal stack state is not reported.

Return Values
Upon successful completion, the sigstack subroutine returns a value of 0. Otherwise, a value of -1 is
returned and the errno global variable is set to indicate the error.
Error Codes

The **sigstack** subroutine is unsuccessful and the signal stack context remains unchanged if the following is true:

**EFAULT** The InStack or OutStack parameter points outside of the address space of the process.

Related Information

The **longjmp** subroutine, **setjmp** subroutine, **sigaction** subroutine, **signal** subroutine, **sigvec** subroutine.

sigsuspend or sigpause Subroutine

**Purpose**

Automatically changes the set of blocked signals and waits for a signal.

**Library**

Standard C Library (**libc.a**)

**Syntax**

```c
#include <signal.h>

int sigsuspend (SignalMask);
const sigset_t *SignalMask;
int sigpause (SignalMask)
int SignalMask;
```

**Description**

The **sigsuspend** subroutine replaces the signal mask of a thread with the set of signals pointed to by the **SignalMask** parameter. It then suspends execution of the thread until a signal is delivered that executes a signal-catching function or terminates the process. The **sigsuspend** subroutine does not allow the **SIGKILL** or **SIGSTOP** signal to be blocked. If a program attempts to block one of these signals, the **sigsuspend** subroutine gives no indication of the error.

If delivery of a signal causes the process to end, the **sigsuspend** subroutine does not return. If delivery of a signal causes a signal-catching function to start, the **sigsuspend** subroutine returns after the signal-catching function returns, with the signal mask restored to the set that existed prior to the **sigsuspend** subroutine.

The **sigsuspend** subroutine sets the signal mask and waits for an unblocked signal as one atomic operation. This means that signals cannot occur between the operations of setting the mask and waiting for a signal. If a program invokes the **sigprocmask** (**SIG_SETMASK**) and **pause** subroutines separately, a signal that occurs between these subroutines might not be noticed by the **pause** subroutine.

In normal usage, a signal is blocked by using the **sigprocmask**(**SIG_BLOCK,...**) subroutine for single-threaded applications, or the **sighandler**(**SIG_BLOCK,...**) subroutine for multi-threaded applications (using the **libpthreads.a** threads library) at the beginning of a critical section. The process/thread then determines whether there is work for it to do. If no work is to be done, the process/thread waits for work by calling the **sigsuspend** subroutine with the mask previously returned by the **sigprocmask** or **sighandler** subroutine.
The `sigpause` subroutine is provided for compatibility with older UNIX systems; its function is a subset of the `sigsuspend` subroutine.

**Parameter**

*SignalMask* Points to a set of signals.

**Return Values**

If a signal is caught by the calling thread and control is returned from the signal handler, the calling thread resumes execution after the `sigsuspend` or `sigpause` subroutine, which always return a value of -1 and set the `errno` global variable to `EINTR`.

**Related Information**

The `pause` subroutine, `sigprocmask` subroutine, `sigaction` or `signal` subroutine, `sigthreadmask` subroutine. (For more details, see the AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs manual.)

### sigthreadmask Subroutine

**Purpose**

Sets the signal mask of a thread.

**Library**

Threads Library (`libpthreads.a`)

**Syntax**

```c
#include <pthread.h>
#include <signal.h>

int sigthreadmask( how, set, old_set )
int how;
const sigset_t *set;
sigset_t *old_set;
```

**Description**

The `sigthreadmask` subroutine is used to examine or change the signal mask of the calling thread. The `sigprocmask` subroutine must not be used in a multi-threaded process.

Typically, the `sigthreadmask(SIG_BLOCK)` subroutine is used to block signals during a critical section of code. The `sigthreadmask(SIG_SETMASK)` subroutine is then used to restore the mask to the previous value returned by the `sigthreadmask(SIG_BLOCK)` subroutine.

If there are any pending unblocked signals after the call to the `sigthreadmask` subroutine, at least one of those signals will be delivered before the `sigthreadmask` subroutine returns.

The `sigthreadmask` subroutine does not allow the `SIGKILL` or `SIGSTOP` signal to be blocked. If a program attempts to block either signal, the `sigthreadmask` subroutine gives no indication of the error.
Note: The pthread.h header file must be the first included file of each source file using the threads library.

Parameters

how  Indicates the manner in which the set is changed. It can have one of the following values:

SIG_BLOCK
   The resulting set is the union of the current set and the signal set pointed to by the set parameter.

SIG_UNBLOCK
   The resulting set is the intersection of the current set and the complement of the signal set pointed to by the set parameter.

SIG_SETMASK
   The resulting set is the signal set pointed to by the set parameter.

set  Specifies the signal set. If the value of the Set parameter is not null, it points to a set of signals to be used to change the currently blocked set. If the value of the Set parameter is null, the value of the How parameter is not significant and the process signal mask is unchanged. Thus, the call can be used to inquire about currently blocked signals.

old_set  If the old_set parameter is not the null value, the signal mask in effect at the time of the call is stored in the spaced pointed to by the old_set parameter.

Return Values

Upon completion, a value of 0 is returned. If the sigthreadmask subroutine fails, the signal mask of the process is unchanged, a value of -1 is returned, and the global variable errno is set to indicate the error.

Error Codes

The sigthreadmask subroutine is unsuccessful if the following is true:

EFAULT   The set or old_set pointers are not in the process address space.
EINVAL   The value of the how parameter is not supported.
EPERM   The calling thread does not have the privilege to change the signal’s mask.

Examples

To set the signal mask to block only the SIGINT signal from delivery, enter:

```c
#include <pthread.h>
#include <signal.h>

int return_value;
 sigset_t newset;
 sigset_t *newset_p;
 .
 newset_p = &newset;
 sigemptyset(newset_p);
 sigaddset(newset_p, SIGINT);
 return_value = sigthreadmask(SIG_SETMASK, newset_p, NULL);
```

Related Information

The kill or killpg subroutine, pthread_kill subroutine, sigaction, sigvec, or signal subroutine, sigpause subroutine, sigpending subroutine, sigsuspend subroutine.
sigtimedwait and sigwaitinfo Subroutine

Purpose
Waits for a signal, and provides a mechanism for retrieving any queued value.

Library
Standard C Library (libc.a)
Threads Library (libpthreads.a)

Syntax
```
#include <signal.h>

int sigtimedwait (set, info, timeout)
const sigset_t *set;
siginfo_t *info;
const struct timespec *timeout;

int sigwaitinfo (set, info)
const sigset_t *set;
siginfo_t *info;
```

Description
The `sigwaitinfo` subroutine selects a pending signal from the set specified by the `set` parameter. If no signal in the `set` parameter is pending at the time of the call, the calling thread is suspended until one or more signals in the `set` parameter become pending or until it is interrupted by an unblocked, caught signal. If the wait was interrupted by an unblocked, caught signal, the subroutines will restart themselves.

The `sigwaitinfo` subroutine is functionally equivalent to the `sigwait` subroutine if the `info` argument is NULL. If the `info` argument is non-NULL, the `sigwaitinfo` subroutine is equivalent to the `sigwait` subroutine, except that the selected signal number is stored in the `si_signo` member, and the cause of the signal is stored in the `si_code` member of the `info` parameter. If any value is queued to the selected signal, the first such queued value is dequeued, and if the `info` argument is non-NULL, the value is stored in the `si_value` member of the `info` parameter. If no further signals are queued for the selected signal, the pending indication for that signal is reset.

The `sigtimedwait` subroutine is equivalent to the `sigwaitinfo` subroutine except that if none of the signals specified by the `set` parameter are pending, the `sigtimedwait` subroutine waits for the time interval referenced by the `timeout` parameter. If the `timespec` structure pointed to by the `timeout` parameter contains a zero value and if none of the signals specified by the `set` parameter are pending, the `sigtimedwait` subroutine returns immediately with an error.

If there are multiple pending signals in the range `SIGRTMIN` to `SIGRTMAX`, the lowest numbered signal in that range will be selected.

Note: All signals in set should have been blocked prior to calling any of the `sigwait` subroutines.

Parameters
- `set` Specifies the pending signals that may be selected.
- `info` Points to a `siginfo_t` in which additional signal information can be returned.
timeout Points to the timespec structure.

Return Values
Upon successful completion, the sigtimedwait and sigwaitinfo subroutines return the selected signal number. If unsuccessful, the sigtimedwait and sigwaitinfo subroutines return -1 and set the errno variable to indicate the error.

Error Codes
The sigtimedwait subroutine will fail if:

EAGAIN No signal specified by the set parameter was generated within the specified timeout period.

The sigtimedwait and sigwaitinfo subroutines may fail if:

EINVAL The set parameter is empty, or contains an invalid, non-catchable, or unsupported signal number.

The sigtimedwait subroutine may also fail when none of the selected signals are pending if:

EINVAL The timeout parameter specified a tv_nsec value less than zero or greater than or equal to 1000 million.

Related Information
“sigqueue Subroutine” on page 221 and “sigwait Subroutine.”

sigwait Subroutine

Purpose
Blocks the calling thread until a specified signal is received.

Library
Threads Library (libpthread.a)

Syntax
#include <usr/include/sys/signal.h>

int sigwait (set, sig)
const sigset_t *set;
it *sig;

Description
The sigwait subroutine blocks the calling thread until one of the signal in the signal set set is received by the thread. sigwait returns an EINVAL error if it attempts to wait on SIGKILL(9), SIGSTOP(17), or SIGWAITING(39–AIX-specific).

The signal can be either sent directly to the thread, using the pthread_kill subroutine, or to the process. In that case, the signal will be delivered to exactly one thread that has not blocked the signal.

Concurrent use of sigaction and sigwait subroutines on the same signal is forbidden.
Parameters

set Specifies the set of signals to wait on.
sig Points to where the received signal number will be stored.

Return Values
Upon successful completion, the received signal number is returned via the sig parameter, and 0 is returned. Otherwise, an error code is returned.

Error Code
The sigwait subroutine is unsuccessful if the following is true:
EINVAL The set parameter contains an invalid or unsupported signal number.

Related Information
The kill subroutine, pthread_kill subroutine, sigaction subroutine, sigthreadmask subroutine.

Signal Management in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

sin, sinf, or sinl Subroutine

Purpose
Computes the sine.

Syntax
#include <math.h>

double sin (x)
  double x;
float sinf (x)
  float x;
long double sinl (x)
  long double x;

Description
The sin, sinf, sinl subroutines compute the sine of the x parameter, measured in radians.

An application wishing to check for error situations should set the errno global variable to zero and call feclearexcept(FE_ALL_EXCEPT) before calling these subroutines. Upon return, if errno is nonzero or fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is nonzero, an error has occurred.

Parameters

x Floating-point value
y Floating-point value
Return Values
Upon successful completion, the sin, sinc, and sinl subroutines return the sine of x.

If x is NaN, a NaN is returned.

If x is ±0, x is returned.

If x is subnormal, a range error may occur and x should be returned.

If x is ±Inf, a domain error occurs, and a NaN is returned.

Error Codes
The sin, sinc, and sinl subroutines lose accuracy when passed a large value for the x parameter. In the sin subroutine, for example, values of x that are greater than pi are argument-reduced by first dividing them by the machine value for 2 * pi, and then using the IEEE remainder of this division in place of x. Since the machine value of pi can only approximate its infinitely precise value, the remainder of x/(2 * pi) becomes less accurate as x becomes larger. Similar loss of accuracy occurs for the sinl subroutine during argument reduction of large arguments.

sin  When the x parameter is extremely large, these functions return 0 when there would be a complete loss of significance. In this case, a message indicating TLOSS error is printed on the standard error output. For less extreme values causing partial loss of significance, a PLOSS error is generated but no message is printed. In both cases, the errno global variable is set to a ERANGE value.

These error-handling procedures may be changed with the matherr subroutine when using the libmsaa.a (-lmsaa) library.

Related Information
The matherr subroutine, sinh, sinhl ("sinh, sinhf, or sinhl Subroutine") subroutines.

Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

128-Bit long double Floating-Point Format in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

math.h in AIX 5L Version 5.3 Files Reference.

sinh, sinhf, or sinhl Subroutine

Purpose
Computes hyperbolic sine.

Syntax
#include <math.h>

double sinh (x)
double x;
float sinh (x)
float x;
long double sinh1 (x)
double x;

Description
The sinh, sinhf, and sinh1 subroutines compute the hyperbolic sine of the x parameter.

An application wishing to check for error situations should set the errno global variable to zero and call feclearexcept(FE_ALL_EXCEPT) before calling these subroutines. Upon return, if errno is nonzero or fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is nonzero, an error has occurred.

Parameters
x Specifies a double-precision floating-point value.

Return Values
Upon successful completion, the sinh, sinhf, and sinh1 subroutines return the hyperbolic sine of x.

If the result would cause an overflow, a range error occurs and ±HUGE_VAL, ±HUGE_VALL, and ±HUGE_VALL (with the same sign as x) is returned as appropriate for the type of the function.

If x is NaN, a NaN is returned.

If x is ±0 orInf, x is returned.

If x is subnormal, a range error may occur and x should be returned.

Error Codes
If the correct value overflows, the sinh, sinhf and sinh1 subroutines return a correctly signed HUGE_VAL, and the errno global variable is set to ERANGE.

These error-handling procedures should be changed with the matherr subroutine when the libmsaa.a (-lmsaa) library is used.

Related Information
asinh, acosh, or atanh Subroutine, feclearexcept Subroutine, fetestexcept Subroutine, and class, _class, finite, isnan, or unordered Subroutines in AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 1.

math.h in AIX 5L Version 5.3 Files Reference.

The matherr subroutine, sin, asin, acos, atan, or atan2 “sin, sinf, or sinl Subroutine” on page 233 subroutine.

Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

128-Bit long double Floating-Point Format in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
sleep, nsleep or usleep Subroutine

Purpose
Suspended a current process from execution.

Library
Standard C Library (libc.a)

Syntax
#include <unistd.h>
unsigned int sleep (Seconds)

#include <sys/time.h>
int nsleep (Rqtp, Rmtp)
struct timestruc_t *Rqtp, *Rmtp;

int usleep (Useconds)
useconds_t Useconds;

Description
The nsleep subroutine is an extended form of the sleep subroutine. The sleep or nsleep subroutines suspend the current process until:

- The time interval specified by the Rqtp parameter elapses.
- A signal is delivered to the calling process that invokes a signal-catching function or terminates the process.
- The process is notified of an event through an event notification function.

The suspension time may be longer than requested due to the scheduling of other activity by the system. Upon return, the location specified by the Rmtp parameter shall be updated to contain the amount of time remaining in the interval, or 0 if the full interval has elapsed.

Parameters
Rqtp Time interval specified for suspension of execution.
Rmtp Specifies the time remaining on the interval timer or 0.
Seconds Specifies time interval in seconds.
Useconds Specifies time interval in microseconds.

Compatibility Interfaces
The sleep and usleep subroutines are provided to ensure compatibility with older versions of the operating system, AT&T System V and BSD systems. They are implemented simply as front-ends to the nsleep subroutine. Programs linking with the libbsd.a library get a BSD compatible version of the sleep subroutine. The return value from the BSD compatible sleep subroutine has no significance and should be ignored.

Return Values
The nsleep, sleep, and usleep subroutines return a value of 0 if the requested time has elapsed.

If the nsleep subroutine returns a value of -1, the notification of a signal or event was received and the Rmtp parameter is updated to the requested time minus the time actually slept (unslept time), and the errno global variable is set.
If the `sleep` subroutine returns because of a 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.

**Error Codes**

If the `nsleep` subroutine fails, a value of -1 is returned and the `errno` global variable is set to one of the following error codes:

- **EINTR**  A signal was caught by the calling process and control has been returned from the signal-catching routine, or the process has been notified of an event through an event notification function.
- **EINVAL**  The `Rqtp` parameter specified a nanosecond value less than zero or greater than or equal to one second.

The `sleep` subroutine is always successful and no return value is reserved to indicate an error.

**Related Information**

The `alarm`, `pause`, `sigaction` subroutine.


---

**sockatmark Subroutine**

**Purpose**

Determines whether a socket is at the out-of-band mark.

**Syntax**

```c
#include <sys/socket.h>

int sockatmark(s)
  int s;
```

**Description**

The `sockatmark` subroutine determines whether the socket specified by the `s` parameter is at the out-of-band data mark. If the protocol for the socket supports out-of-band data by marking the stream with an out-of-band data mark, the `sockatmark` subroutine returns a 1 when all data preceding the mark has been read and the out-of-band data mark is the first element in the receive queue. The `sockatmark` subroutine does not remove the mark from the stream.

The use of this subroutine between receive operations allows an application to determine which received data precedes the out-of-band data and which follows the out-of-band data. There is an inherent race condition in the use of this function. On an empty receive queue, the current read of the location might well be at the mark’, but the system has no way of knowing that the next data segment that will arrive from the network will carry the mark, and `sockatmark` will return false The next read operation will silently consume the mark. Because of this, the `sockatmark` subroutine can only be used reliably when the application already knows that the out-of-band data has been seen by the system or that it is known that there is data waiting to be read at the socket.

**Parameters**

- `s`  Specifies the socket to be checked.
Return Values
Upon successful completion, the `sockatmark` subroutine returns a value indicating whether the socket is at an out-of-band data mark. If the protocol has marked the data stream and all data preceding the mark has been read, the return value is 1. If there is no mark, or if data precedes the mark in the receive queue, the `sockatmark` subroutine returns a 0. Otherwise, it returns a value of -1 and sets the `errno` global variable to indicate the error.

Error Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBADF</td>
<td>The s parameter is not a valid file descriptor.</td>
</tr>
<tr>
<td>ENOTTY</td>
<td>The s parameter does not specify a descriptor for a socket.</td>
</tr>
</tbody>
</table>

SpmiAddSetHot Subroutine

Purpose
Adds a set of peer statistics values to a hotset.

Library
SPMI Library (libSpmi.a)

Syntax
```
#include <sys/Spmidef.h>
struct SpmiHotVals *SpmiAddSetHot(HotSet, StatName, GrandParent, maxresp, threshold, frequency, feed_type, except_type, severity, trap_no)
```

Description
The `SpmiAddSetHot` subroutine adds a set of peer statistics to a hotset. The `SpmiHotSet` structure that provides the anchor point to the set must exist before the `SpmiAddSetHot` subroutine call can succeed.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

Parameters

**HotSet**
Specifies a pointer to a valid structure of type `SpmiHotSet` as created by the `SpmiCreateHotSet` subroutine call.

**StatName**
Specifies the name of the statistic within the subcontexts (peer contexts) of the context identified by the `GrandParent` parameter.
GrandParent

Specifies a valid SpmiCxHdl handle as obtained by another subroutine call. The handle must identify a context with at least one subcontext, which contains the statistic identified by the StatName parameter. If the context specified is one of the RTime contexts, no subcontext need to exist at the time the SpmiAddSetHot subroutine call is issued; the presence of the metric identified by the StatName parameter is checked against the context class description.

If the context specified has or may have multiple levels of instantiable context below it (such as the FS and RTime/ARM contexts), the metric is only searched for at the lowest context level. The SpmiHotSet created is a pseudo hotvals structure used to link together a peer group of SpmiHotVals structures, which are created under the covers, one for each subcontext of the GrandParent context. In the case of RTime/ARM, if additional contexts are later added under the GrandParent contexts, additional hotsets are added to the peer group. This is transparent to the application program, except that the SpmiFirstHot, SpmiNextHot, and SpmiNextHotItem subroutine calls will return the peer group SpmiHotVals pointer rather than the pointer to the pseudo structure.

Note that specifying a specific volume group context (such as FS/rootvg) or a specific application context (such as RTime/ARN/armpeek) is still valid and won’t involve creation of pseudo SpmiHotVals structures.

maxresp

Must be non-zero if exceps_type specifies that exceptions or SNMP traps must be generated. If specified as zero, indicates that all SpmiHotItems that meet the criteria specified by threshold must be returned, up-to a maximum of maxresp items. If both exceptions/traps and feeds are requested, the maxresp value is used to cap the number of exceptions/alerts as well as the number of items returned. If feed_type is specified as SiHotAlways, the maxresp parameter is still used to return at most maxresp items.

Where the GrandParent argument specifies a context that has multiple levels of instantiable contexts below it, the maxresp is applied to each of the lowest level contexts above the the actual peer contexts at a time. For example, if the GrandParent context is FS (file systems) and the system has three volume groups, then a maxresp value of 2 could cause up to a maximum of 2 x 3 = 6 responses to be generated.

threshold

Must be non-zero if exceps_type specifies that exceptions or SNMP traps must be generated. If specified as zero, indicates that all values read qualify to be returned in feeds. The value specified is compared to the data value read for each peer statistic. If the data value exceeds the threshold, it qualifies to be returned as an SpmiHotItems element in the SpmiHotVals structure. If the threshold is specified as a negative value, the value qualifies if it is lower than the numeric value of threshold. If feed_type is specified as SiHotAlways, the threshold value is ignored for feeds. For peer statistics of type SiCounter, the threshold must be specified as a rate per second; for SiQuantity statistics the threshold is specified as a level.

frequency

Must be non-zero if exceps_type specifies that exceptions or SNMP traps must be generated. Ignored for feeds. Specifies the minimum number of minutes that must expire between any two exceptions/traps generated from this SpmiHotVals structure. This value must be specified as no less than 5 minutes.

feed_type

Specifies if feeds of SpmiHotItems should be returned for this SpmiHotVals structure. The following values are valid:

SiHotNoFeed

No feeds should be generated
SiHotThreshold
Feeds are controlled by threshold.

SiHotAlways
All values, up-to a maximum of maxresp must be returned as feeds.

excp_type
Controls the generation of exception data packets and/or the generation of SNMP Traps from xmservd. Note that these types of packets and traps can only actually be sent if xmservd is running. Because of this, exception packets and SNMP traps are only generated as long as xmservd is active. Traps can only be generated on AIX systems. The conditions for generating exceptions and traps are controlled by the threshold and frequency parameters. The following values are valid for excp_type:

SiNoHotException
Generate neither exceptions not traps.

SiHotException
Generate exceptions but not traps.

SiHotTrap
Generate SNMP traps but not exceptions.

SiHotBoth
Generate both exceptions and SNMP traps.

severity
Required to be positive and greater than zero if exceptions are generated, otherwise specify as zero. Used to assign a severity code to the exception for display by exmon.

trap_no
Required to be positive and greater than zero if SNMP traps are generated, otherwise specify as zero. Used to assign the trap number in the generated SNMP trap.

Return Values
The SpmiAddSetHot subroutine returns a pointer to a structure of type SpmiHotVals if successful. If unsuccessful, the subroutine returns a NULL value.

Programming Notes
The SpmiAddSetHot functions in a straight forward manner and as described previously in all cases where the GrandParent context is a context that has only one level of instantiable contexts below it. This covers most context types such as CPU, Disk, LAN, etc. In a few cases, currently only the FS (file system) and RTime/ARM (application response) contexts, the SPMI works by creating pseudo-hotvals structures that effectively expand the hotset. These pseudo-hotvals structures are created either at the time the SpmiAddSetHot call is issued or when new subcontexts are created for a context that’s already the GrandParent of a hotvals peer set. For example:

When a peer set is created for RTime/ARM, maybe only a few or no subcontexts of this context exists. If two applications were defined at this point, say checking and savings, one valsset would be created for the RTime/ARM context and a pseudo-valsset for each of RTime/ARM/checking and RTime/ARM/savings. As new applications are added to the RTime/ARM contexts, new pseudo-valssets are automatically added to the hotset.

Pseudo-valssets represent an implementation convenience and also helps minimize the impact of retrieving and presenting data for hotsets. As far as the caller of the RSiGetHotItem subroutine call is concerned, it is completely transparent. All this caller will ever see is the real hotvals structure. That is not the case for callers of SpmiFirstHot, SpmiNextHot, and SpmiNextHotItem. All of these subroutines will return pseudo-valssets and the calling program should be prepared to handle this.
Error Codes
All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char SpmiErrmsg[];
- extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrmsg variable contains text, in English, explaining the cause of the error. See the List of SPMI Error Codes for more information.

Files
/usr/include/sys/Spmidef.h
Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

SpmiCreateHotSet

Purpose
Creates an empty hotset.

Library
SPMI Library (libSpmi.a)

Syntax
#include sys/Spmidef.h
struct SpmiHotSet *SpmiCreateHotSet()

Description
The SpmiCreateHotSet subroutine creates an empty hotset and returns a pointer to an SpmiHotSet structure. This structure provides the anchor point for a hotset and must exist before the SpmiAddSetHot subroutine can be successfully called.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

Return Values
The SpmiCreateHotSet subroutine returns a pointer to a structure of type SpmiHotSet if successful. If unsuccessful, the subroutine returns a NULL value.

Error Codes
All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char SpmiErrmsg[];
- extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file.
in the `sys/Spmidef.h` file, and the `SpmiErrmsg` variable contains text, in English, explaining the cause of the error. See the [List of SPMI Error Codes](#) for more information.

### Files

`/usr/include/sys/Spmidef.h` declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

### Related Information

For related information, see:

- **“SpmiDelSetHot Subroutine” on page 247**
- **“SpmiFreeHotSet Subroutine” on page 255**
- **“SpmiAddSetHot Subroutine” on page 238**
- [Understanding SPMI Data Areas](#)

---

## SpmiCreateStatSet Subroutine

### Purpose

Creates an empty set of statistics.

### Library

SPMI Library (`libSpmi.a`)

### Syntax

```c
#include sys/Spmidef.h
struct SpmiStatSet *SpmiCreateStatSet();
```

### Description

The `SpmiCreateStatSet` subroutine creates an empty set of statistics and returns a pointer to an `SpmiStatSet` structure.

The `SpmiStatSet` structure provides the anchor point to a set of statistics and must exist before the `SpmiPathAddSetStat` subroutine can be successfully called.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

### Return Values

The `SpmiCreateStatSet` subroutine returns a pointer to a structure of type `SpmiStatSet` if successful. If unsuccessful, the subroutine returns a NULL value.

### Error Codes

All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- `extern char SpmiErrmsg[];
- `extern int SpmiErrno;

If the subroutine returns without an error, the `SpmiErrno` variable is set to 0 and the `SpmiErrmsg` character array is empty. If an error is detected, the `SpmiErrno` variable returns an error code, as defined in the SPMI Library (`libSpmi.a`).
in the `sys/Spmidef.h` file, and the `SpmiErrmsg` variable contains text, in English, explaining the cause of the error. See the [List of SPMI Error Codes](#) for more information.

**Files**

`/usr/include/sys/Spmidef.h` Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

**Related Information**

For related information, see:

- "SpmiDelSetStat Subroutine" on page 248
- "SpmiFreeStatSet Subroutine" on page 256
- "SpmiPathAddSetStat Subroutine" on page 274
- [Understanding SPMI Data Areas](#)

### SpmiDdsAddCx Subroutine

**Purpose**

Adds a volatile context to the contexts defined by an application.

**Library**

SPMI Library (libSpmi.a)

**Syntax**

```
#include <sys/Spmidef.h>

char *SpmiDdsAddCx(Ix, Path, Descr, Asnno)
ushort Ix;
char *Path, *Descr;
int Asnno;
```

**Description**

The `SpmiDdsAddCx` subroutine uses the shared memory area to inform the SPMI that a context is available to be added to the context hierarchy, moves a copy of the context to shared memory, and allocates memory for the data area.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

**Parameters**

- **Ix**

  Specifies the element number of the added context in the table of dynamic contexts. No context can be added if the table of dynamic contexts has not been defined in the `SpmiDdsInit` subroutine call. The first element of the table is element number 0.

- **Path**

  Specifies the full path name of the context to be added. If the context is not at the top-level, the parent context must already exist.

- **Descr**

  Provides the description of the context to be added as it will be presented to data consumers.
Asnno

Specifies the ASN.1 number to be assigned to the new context. All subcontexts on the same level as the new context must have unique ASN.1 numbers. Typically, each time the SpmiDdsAddCx subroutine adds a subcontext to the same parent context, the Asnno parameter is incremented. See [Making Dynamic Data-Supplier Statistics Unique](#) for more information about ASN.1 numbers.

Return Values

If successful, the SpmiDdsAddCx subroutine returns the address of the shared memory data area. If an error occurs, an error text is placed in the external SpmiErrmsg character array, and the subroutine returns a NULL value.

Error Codes

All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char SpmiErrmsg[];
- extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrmsg variable contains text, in English, explaining the cause of the error. See the [List of SPMI Error Codes](#) for more information.

Files

/usr/include/sys/Spmidef.h

Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

Related Information

For related information, see:

- "SpmiDdsDelCx Subroutine"
- "SpmiDdsInit Subroutine" on page 245

SpmiDdsDelCx Subroutine

Purpose

Deletes a volatile context.

Library

SPMI Library (libSpmi.a)

Syntax

```c
#include sys/Spmidef.h
int SpmiDdsDelCx/Area
char *Area;
```

Description

The SpmiDdsDelCx subroutine informs the SPMI that a previously added, volatile context should be deleted.
If the SPMI has not detected that the context to delete was previously added dynamically, the **SpmiDdsDelCx** subroutine removes the context from the list of to-be-added contexts and returns the allocated shared memory to the free list. Otherwise, the **SpmiDdsDelCx** subroutine indicates to the SPMI that a context and its associated statistics must be removed from the context hierarchy and any allocated shared memory must be returned to the free list.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

**Parameters**

**Area**

Specifies the address of the previously allocated shared memory data area as returned by an **SpmiDdsAddCx** subroutine call.

**Return Values**

If successful, the **SpmiDdsDelCx** subroutine returns a value of 0. If an error occurs, an error text is placed in the external **SpmiErrmsg** character array, and the subroutine returns a nonzero value.

**Error Codes**

All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char SpmiErrmsg[];
- extern int SpmiErrno;

If the subroutine returns without an error, the **SpmiErrno** variable is set to 0 and the **SpmiErrmsg** character array is empty. If an error is detected, the **SpmiErrno** variable returns an error code, as defined in the **sys/Spmidef.h** file, and the **SpmiErrmsg** variable contains text, in English, explaining the cause of the error. See the **List of SPMI Error Codes** for more information.

**Files**

/usr/include/sys/Spmidef.h Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

**Related Information**

For related information, see:

- “**SpmiDdsAddCx** Subroutine” on page 243
- “**SpmiDdsInit** Subroutine”
- Understanding SPMI Data Areas

---

**SpmiDdsInit Subroutine**

**Purpose**

- Establishes a program as a dynamic data-supplier (DDS) program.

**Library**

SPMI Library (libSpmi.a)
Syntax
```c
#include sys/Spmidef.h

SpmiShare *SpmiDdsInit(CxTab, CxCnt, IxTab, IxCnt, FileName)

cx_create *CxTab, *IxTab;
int CxCnt, IxCnt;
char *FileName;
```

Description
The SpmiDdsInit subroutine establishes a program as a dynamic data-supplier (DDS) program. To do so, the SpmiDdsInit subroutine:
1. Determines the size of the shared memory required and creates a shared memory segment of that size.
2. Moves all static contexts and all statistics referenced by those contexts to the shared memory.
3. Calls the SPMI and requests it to add all of the DDS static contexts to the context tree.

Notes:
1. The SpmiDdsInit subroutine issues an Spmilnit subroutine call if the application program has not issued one.
2. If the calling program uses shared memory for other purposes, including memory mapping of files, the SpmiDdsInit or the Spmilnit subroutine call must be issued before access is established to other shared memory areas.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

Parameters

**CxTab**
Specifies a pointer to the table of nonvolatile contexts to be added.

**CxCnt**
Specifies the number of elements in the table of nonvolatile contexts. Use the CX_L macro to find this value.

**IxTab**
Specifies a pointer to the table of volatile contexts the program may want to add later. If no contexts are defined, specify NULL.

**IxCnt**
Specifies the number of elements in the table of volatile contexts. Use the CX_L macro to find this value. If no contexts are defined, specify 0.

**FileName**
Specifies the fully qualified path and file name to use when creating the shared memory segment. At execution time, if the file exists, the process running the DDS must be able to write to the file. Otherwise, the SpmiDdsInit subroutine call does not succeed. If the file does not exist, it is created. If the file cannot be created, the subroutine returns an error. If the file name includes directories that do not exist, the subroutine returns an error.

For non-AIX systems, a sixth argument is required to inform the SPMI how much memory to allocate in the DDS shared memory segment. This is not required for AIX systems because facilities exist to expand a memory allocation in shared memory. The sixth argument is:

**size**
Size in bytes of the shared memory area to allocate for the DDS program. This parameter is of type int.

Return Values
If successful, the `SpmiDdsInit` subroutine returns the address of the shared memory control area. If an error occurs, an error text is placed in the external `SpmiErrmsg` character array, and the subroutine returns a NULL value.

Error Codes
All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- `extern char SpmiErrmsg[];`
- `extern int SpmiErrno;`

If the subroutine returns without an error, the `SpmiErrno` variable is set to 0 and the `SpmiErrmsg` character array is empty. If an error is detected, the `SpmiErrno` variable returns an error code, as defined in the `sys/Spmidef.h` file, and the `SpmiErrmsg` variable contains text, in English, explaining the cause of the error. See the List of SPMI Error Codes for more information.

Files

/`usr/include/sys/Spmidef.h` Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

Related Information
For related information, see:

- "SpmiExit Subroutine" on page 250
- "SpmiInit Subroutine" on page 263
- Understanding SPMI Data Areas

SpmiDelSetHot Subroutine

Purpose
Removes a single set of peer statistics from a hotset.

Library
SPMI Library (libSpmi.a)

Syntax
```c
#include sys/Spmidef.h
int SpmiDelSetHot(HotSet, HotVal)
struct SpmiHotSet *HotSet;
struct SpmiHotVals *HotVal;
```

Description
The `SpmiDelSetHot` subroutine removes a single set of peer statistics, identified by the `HotVal` parameter, from a hotset, identified by the `HotSet` parameter.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.
Parameters

HotSet
Specifies a pointer to a valid structure of type SpmiHotSet as created by the "SpmiCreateHotSet" subroutine call.

HotVal
Specifies a pointer to a valid structure of type SpmiHotVals as created by the "SpmiAddSetHot Subroutine" on page 238 subroutine call. You cannot specify an SpmiHotVals that was internally generated by the SPMI library code as described under the GrandParent parameter to SpmiAddSetHot.

Return Values
The SpmiDelSetHot subroutine returns a value of 0 if successful. If unsuccessful, the subroutine returns a nonzero value.

Error Codes
All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char SpmiErrmsg[];
- extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrmsg variable contains text, in English, explaining the cause of the error. See the List of SPMI Error Codes for more information.

Files
/usr/include/sys/Spmidef.h Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

Related Information
For related information, see:
- "SpmiCreateHotSet" on page 241
- "SpmiFreeHotSet Subroutine" on page 255
- "SpmiAddSetHot Subroutine" on page 238
- Understanding SPMI Data Areas

SpmiDelSetStat Subroutine

Purpose
Removes a single statistic from a set of statistics.

Library
SPMI Library (libSpmi.a)

Syntax
#include sys/Spmidef.h
Description
The `SpmiDelSetStat` subroutine removes a single statistic, identified by the `StatVal` parameter, from a set of statistics, identified by the `StatSet` parameter.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

Parameters

`StatSet`
Specifies a pointer to a valid structure of type `SpmiStatSet` as created by the "SpmiCreateStatSet Subroutine" on page 242 subroutine call.

`StatVal`
Specifies a pointer to a valid structure of type `SpmiStatVals` as created by the "SpmiPathAddSetStat Subroutine" on page 274 subroutine call.

Return Values
The `SpmiDelSetStat` subroutine returns a value of 0 if successful. If unsuccessful, the subroutine returns a nonzero value.

Error Codes
All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- `extern char SpmiErrmsg[];`
- `extern int SpmiErrno;`

If the subroutine returns without an error, the `SpmiErerrno` variable is set to 0 and the `SpmiErrmsg` character array is empty. If an error is detected, the `SpmiErerrno` variable returns an error code, as defined in the `sys/Spmidef.h` file, and the `SpmiErrmsg` variable contains text, in English, explaining the cause of the error. See the `List of SPMI Error Codes` for more information.

Files

`/usr/include/sys/Spmidef.h` Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

Related Information
For related information, see:

- "SpmiCreateStatSet Subroutine" on page 242
- "SpmiFreeStatSet Subroutine" on page 256
- "SpmiPathAddSetStat Subroutine" on page 274
- `Understanding SPMI Data Areas`
**SpmiExit Subroutine**

**Purpose**
Terminates a dynamic data supplier (DDS) or local data consumer program’s association with the SPMI, and releases allocated memory.

**Library**
SPMI Library (*libSpmi.a*)

**Syntax**

```c
#include sys/Spmidef.h
void SpmiExit()
```

**Description**
A successful "SpmiInit Subroutine" on page 263 or "SpmiDdsInit Subroutine" on page 245 call allocates shared memory. Therefore, a Dynamic Data Supplier (DDS) program that has issued a successful SpmiInit or SpmiDdsInit subroutine call should issue an SpmiExit subroutine call before the program exits the SPMI. Allocated memory is not released until the program issues an SpmiExit subroutine call.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

**Files**

```
/usr/include/sys/Spmidef.h
```

Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

**Related Information**
For related information, see:
- "SpmiInit Subroutine" on page 263
- "SpmiDdsInit Subroutine" on page 245

---

**SpmiFirstCx Subroutine**

**Purpose**
Locates the first subcontext of a context.

**Library**
SPMI Library (*libSpmi.a*)

**Syntax**

```c
#include sys/Spmidef.h
struct SpmiCxLink *SpmiFirstCx(CxHandle)
SpmiCxHdl CxHandle;
```

**Description**
The SpmiFirstCx subroutine locates the first subcontext of a context. The subroutine returns a NULL value if no subcontexts are found.
The structure pointed to by the returned pointer contains a handle to access the contents of the corresponding \texttt{SpmiCx} structure through the \texttt{SpmiGetCx} subroutine call.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

**Parameters**

\textbf{CxHandle}

Specifies a valid \texttt{SpmiCxHdl} handle as obtained by another subroutine call.

**Return Values**

The \texttt{SpmiFirstCx} subroutine returns a pointer to an \texttt{SpmiCxLink} structure if successful. If unsuccessful, the subroutine returns a NULL value.

**Error Codes**

All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- \texttt{extern char SpmiErrmsg[];}
- \texttt{extern int SpmiErrno;}

If the subroutine returns without an error, the \texttt{SpmiErrno} variable is set to 0 and the \texttt{SpmiErrmsg} character array is empty. If an error is detected, the \texttt{SpmiErrno} variable returns an error code, as defined in the \texttt{sys/Spmidef.h} file, and the \texttt{SpmiErrmsg} variable contains text, in English, explaining the cause of the error. See the \texttt{List of SPMI Error Codes} for more information.

**Files**

\texttt{/usr/include/sys/Spmidef.h}

Declarations the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

**Related Information**

For related information, see:

- "\texttt{SpmiGetCx Subroutine}" on page 257
- "\texttt{SpmiNextCx Subroutine}" on page 266
- \texttt{Understanding SPMI Data Areas}
- \texttt{Understanding the SPMI Data Hierarchy}

**\texttt{SpmiFirstHot} Subroutine**

**Purpose**

Locates the first of the sets of peer statistics belonging to a hotset.

**Library**

SPMI Library (libSpmi.a)

**Syntax**

\texttt{#include sys/Spmidef.h}

\texttt{struct SpmiHotVals *SpmiFirstHot(HotSet)}

\texttt{struct SpmiHotSet HotSet;
Description
The SpmiFirstHot subroutine locates the first of the SpmiHotVals structures belonging to the specified SpmiHotSet. Using the returned pointer, the SpmiHotSet can then either be decoded directly by the calling program, or it can be used to specify the starting point for a subsequent SpmiNextHotItem subroutine call. The SpmiFirstHot subroutine should only be executed after a successful call to the SpmiGetHotSet subroutine.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

Parameters
HotSet
Specifies a valid SpmiHotSet structure as obtained by another subroutine call.

Return Values
The SpmiFirstHot subroutine returns a pointer to a structure of type SpmiHotVals structure if successful. If unsuccessful, the subroutine returns a NULL value. A returned pointer may refer to a pseudo-hotvals structure as described in the SpmiAddSetHot subroutine.

Error Codes
All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:
- extern char SpmiErrMsg[];
- extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrMsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrMsg variable contains text, in English, explaining the cause of the error. See the List of SPMI Error Codes for more information.

Files
/usr/include/sys/Spmidef.h
Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

Related Information
For related information, see:
- “SpmiCreateHotSet” on page 241
- “SpmiAddSetHot Subroutine” on page 238
- “SpmiNextHot Subroutine” on page 267
- “SpmiNextHotItem Subroutine” on page 268
- Understanding SPMI Data Areas
- Understanding the SPMI Data Hierarchy

SpmiFirstStat Subroutine
Purpose
Locates the first of the statistics belonging to a context.
Library
SPMI Library (libSpmi.a)

Syntax
#include sys/Spmidef.h
struct SpmiStatLink *SpmiFirstStat(CxHandle)
SpmiCxHdl CxHandle;

Description
The SpmiFirstStat subroutine locates the first of the statistics belonging to a context. The subroutine returns a NULL value if no statistics are found.

The structure pointed to by the returned pointer contains a handle to access the contents of the corresponding SpmiStat structure through the "SpmiGetStat Subroutine" on page 259 call.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

Parameters
CxHandle

   Specifies a valid SpmiCxHdl handle as obtained by another subroutine call.

Return Values
The SpmiFirstStat subroutine returns a pointer to a structure of type SpmiStatLink if successful. If unsuccessful, the subroutine returns a NULL value.

Error Codes
All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

   • extern char SpmiErrmsg[];
   • extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrmsg variable contains text, in English, explaining the cause of the error. See the List of SPMI Error Codes for more information.

Files
/usr/include/sys/Spmidef.h

Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

Related Information
For related information, see:

   • "SpmiGetStat Subroutine" on page 259
   • "SpmiNextStat Subroutine" on page 270
   • Understanding SPMI Data Areas
   • Understanding the SPMI Data Hierarchy
SpmiFirstVals Subroutine

Purpose
Returns a pointer to the first SpmiStatVals structure belonging to a set of statistics.

Library
SPMI Library (libSpmi.a)

Syntax
#include sys/Spmidef.h
struct SpmiStatVals *SpmiFirstVals(StatSet)
struct SpmiStatSet *StatSet;

Description
The SpmiFirstVals subroutine returns a pointer to the first SpmiStatVals structure belonging to the set of statistics identified by the StatSet parameter. SpmiStatVals structures are accessed in reverse order so the last statistic added to the set of statistics is the first one returned. This subroutine call should only be issued after an SpmiGetStatSet subroutine has been issued against the statset.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

Parameters
StatSet
Specifies a pointer to a valid structure of type SpmiStatSet as created by the SpmiCreateStatSet subroutine call.

Return Values
The SpmiFirstVals subroutine returns a pointer to an SpmiStatVals structure if successful. If unsuccessful, the subroutine returns a NULL value.

Error Codes
All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char SpmiErrmsg[];
- extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrmsg variable contains text, in English, explaining the cause of the error. See the List of SPMI Error Codes for more information.

Files
/usr/include/sys/Spmidef.h
Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

254 Technical Reference, Volume 2: Base Operating System and Extensions
SpmiFreeHotSet Subroutine

Purpose
Erases a hotset.

Library
SPMI Library (libSpmi.a)

Syntax
#include sys/Spmidef.h
int SpmiFreeHotSet(HotSet)
struct SpmiHotSet *HotSet;

Description
The SpmiFreeHotSet subroutine erases the hotset identified by the HotSet parameter. All SpmiHotVals structures chained off the SpmiHotSet structure are deleted before the set itself is deleted.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

Parameters
HotSet
   Specifies a pointer to a valid structure of type SpmiHotSet as created by the SpmiCreateHotSet subroutine call.

Return Values
The SpmiFreeHotSet subroutine returns a value of 0 if successful. If unsuccessful, the subroutine returns a nonzero value.

Error Codes
All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:
   • extern char SpmiErrmsg[];
   • extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrmsg variable contains text, in English, explaining the cause of the error. See the List of SPMI Error Codes for more information.
Files
/usr/include/sys/Spmidef.h

Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

Related Information
For related information, see:

- “SpmiCreateHotSet” on page 241
- “SpmiDelSetHot Subroutine” on page 247
- “SpmiAddSetHot Subroutine” on page 238
- Understanding SPMI Data Areas

SpmiFreeStatSet Subroutine

Purpose
Erases a set of statistics.

Library
SPMI Library (libSpmi.a)

Syntax
#include sys/Spmidef.h
int SpmiFreeStatSet(StatSet)
struct SpmiStatSet *StatSet;

Description
The SpmiFreeStatSet subroutine erases the set of statistics identified by the StatSet parameter. All SpmiStatVals structures chained off the SpmiStatSet structure are deleted before the set itself is deleted.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

Parameters
StatSet
Specifies a pointer to a valid structure of type SpmiStatSet as created by the SpmiCreateStatSet subroutine call.

Return Values
The SpmiFreeStatSet subroutine returns a value of 0 if successful. If unsuccessful, the subroutine returns a nonzero value.

Error Codes
All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char SpmiErrmsg[];
- extern int SpmiErrno;
If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrmsg variable contains text, in English, explaining the cause of the error. See the List of SPMI Error Codes for more information.

Files
/usr/include/sys/Spmidef.h

Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

Related Information
For related information, see:
- “SpmiCreateStatSet Subroutine” on page 242
- “SpmiDelSetStat Subroutine” on page 248
- “SpmiPathAddSetStat Subroutine” on page 274
- Understanding SPMI Data Areas

SpmiGetCx Subroutine

Purpose
Returns a pointer to the SpmiCx structure corresponding to a specified context handle.

Library
SPMI Library (libSpmi.a)

Syntax
#include <sys/Spmidef.h>
struct SpmiCx *SpmiGetCx(CxHandle)
SpmiCxHdl CxHandle;

Description
The SpmiGetCx subroutine returns a pointer to the SpmiCx structure corresponding to the context handle identified by the CxHandle parameter.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

Parameters
CxHandle
Specifies a valid SpmiCxHdl handle as obtained by another subroutine call.

Return Values
The SpmiGetCx subroutine returns a a pointer to an SpmiCx data structure if successful. If unsuccessful, the subroutine returns NULL.

Error Codes
All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:
If the subroutine returns without an error, the `SpmiErrno` variable is set to 0 and the `SpmiErrmsg` character array is empty. If an error is detected, the `SpmiErrno` variable returns an error code, as defined in the `sys/Spmidef.h` file, and the `SpmiErrmsg` variable contains text, in English, explaining the cause of the error. See the [List of SPMI Error Codes](#) for more information.

### Files

```
/usr/include/sys/Spmidef.h
```

Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

### Related Information

For related information, see:

- "SpmiFirstCx Subroutine" on page 250
- "SpmiNextCx Subroutine" on page 266
- [Understanding SPMI Data Areas](#)
- [Understanding the SPMI Data Hierarchy](#)

### SpmiGetHotSet Subroutine

#### Purpose

Requests the SPMI to read the data values for all sets of peer statistics belonging to a specified `SpmiHotSet`.

#### Library

SPMI Library (`libSpmi.a`)

#### Syntax

```
#include sys/Spmidef.h

int SpmiGetHotSet(HotSet, Force);
struct SpmiHotSet *HotSet;
boolean Force;
```

#### Description

The `SpmiGetHotSet` subroutine requests the SPMI to read the data values for all peer sets of statistics belonging to the `SpmiHotSet` identified by the `HotSet` parameter. The `Force` parameter is used to force the data values to be refreshed from their source.

The `Force` parameter works by resetting a switch held internally in the SPMI for all `SpmiStatVals` and `SpmiHotVals` structures, regardless of the `SpmiStatSet` and `SpmiHotSet` to which they belong. Whenever the data value for a peer statistic is requested, this switch is checked. If the switch is set, the SPMI reads the latest data value from the original data source. If the switch is not set, the SPMI reads the data value stored in the `SpmiHotVals` structure. This mechanism allows a program to synchronize and minimize the number of times values are retrieved from the source. One method programs can use is to ensure the force request is not issued more than once per elapsed amount of time.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.
Parameters

HotSet

Specifies a pointer to a valid structure of type SpmiHotSet as created by the [SpmiCreateHotSet] subroutine call.

Force

If set to true, forces a refresh from the original source before the SPMI reads the data values for the set. If set to false, causes the SPMI to read the data values as they were previously retrieved from the data source.

When the force argument is set true, the effect is that of marking all statistics known by the SPMI as obsolete, which causes the SPMI to refresh all requested statistics from kernel memory or other sources. As each statistic is refreshed, the obsolete mark is reset. Statistics that are not part of the HotSet specified in the subroutine call remain marked as obsolete. Therefore, if an application repetitively issues a series of, SpmiGetHotSet and SpmiGetStatSet subroutine calls for multiple hotsets and statsets, each time, only the first such call need set the force argument to true.

Return Values

The SpmiGetHotSet subroutine returns a value of 0 if successful. If unsuccessful, the subroutine returns a nonzero value.

Error Codes

All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char SpmiErrmsg[];
- extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrmsg variable contains text, in English, explaining the cause of the error. See the [List of SPMI Error Codes] for more information.

Files

/usr/include/sys/Spmidef.h

Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

Related Information

For related information, see:

- "SpmiCreateHotSet" on page 241
- "SpmiAddSetHot Subroutine" on page 238

SpmiGetStat Subroutine

Purpose

Returns a pointer to the SpmiStat structure corresponding to a specified statistic handle.

Library

SPMI Library (libSpmi.a)
Syntax
#include sys/Spmidef.h
struct SpmiStat *SpmiGetStat(StatHandle)
SpmiStatHdl StatHandle;

Description
The SpmiGetStat subroutine returns a pointer to the SpmiStat structure corresponding to the statistic handle identified by the StatHandle parameter.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

Parameters
StatHandle

Specifies a valid SpmiStatHdl handle as obtained by another subroutine call.

Return Values
The SpmiGetStat subroutine returns a pointer to a structure of type SpmiStat if successful. If unsuccessful, the subroutine returns a NULL value.

Error Codes
All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:
- extern char SpmiErrmsg[];
- extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrmsg variable contains text, in English, explaining the cause of the error. See the List of SPMI Error Codes for more information.

Files
/usr/include/sys/Spmidef.h

Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

Related Information
For related information, see:
- “SpmiFirstStat Subroutine” on page 252
- “SpmiNextStat Subroutine” on page 270
- Understanding SPMI Data Areas
- Understanding the SPMI Data Hierarchy
SpmiGetStatSet Subroutine

Purpose
Requests the SPMI to read the data values for all statistics belonging to a specified set.

Library
SPMI Library (libSpmi.a)

Syntax
```c
#include sys/Spmidef.h
int SpmiGetStatSet(StatSet, Force);
struct SpmiStatSet *StatSet;
boolean Force;
```

Description
The `SpmiGetStatSet` subroutine requests the SPMI to read the data values for all statistics belonging to the `SpmiStatSet` identified by the `StatSet` parameter. The `Force` parameter is used to force the data values to be refreshed from their source.

The `Force` parameter works by resetting a switch held internally in the SPMI for all `SpmiStatVals` and `SpmiHotVals` structures, regardless of the `SpmiStatSets` and `SpmiHotSets` to which they belong. Whenever the data value for a statistic is requested, this switch is checked. If the switch is set, the SPMI reads the latest data value from the original data source. If the switch is not set, the SPMI reads the data value stored for the `SpmiStatVals` structure. This mechanism allows a program to synchronize and minimize the number of times values are retrieved from the source. One method is to ensure the force request is not issued more than once per elapsed amount of time.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

Parameters
- **StatSet**
  Specifies a pointer to a valid structure of type `SpmiStatSet` as created by the `SpmiCreateStatSet` subroutine call.

- **Force**
  If set to true, forces a refresh from the original source before the SPMI reads the data values for the set. If set to false, causes the SPMI to read the data values as they were previously retrieved from the data source.

  When the force argument is set true, the effect is that of marking all statistics known by the SPMI as obsolete, which causes the SPMI to refresh all requested statistics from kernel memory or other sources. As each statistic is refreshed, the obsolete mark is reset. Statistics that are not part of the `StatSet` specified in the subroutine call remain marked as obsolete. Therefore, if an application repetitively issues the `SpmiGetStatSet` and `SpmiGetHotSet` subroutine calls for multiple statsets and hotsets, each time, only the first such call need set the force argument to true.

Return Values
The `SpmiGetStatSet` subroutine returns a value of 0 if successful. If unsuccessful, the subroutine returns a nonzero value.
Error Codes

All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char SpmiErrmsg[];
- extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrmsg variable contains text, in English, explaining the cause of the error. See the List of SPMI Error Codes for more information.

Files

/usr/include/sys/Spmidef.h

Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

Related Information

For related information, see:

- “SpmiCreateStatSet Subroutine” on page 242
- “SpmiPathAddSetStat Subroutine” on page 274

SpmiGetValue Subroutine

Purpose

Returns a decoded value based on the type of data value extracted from the data field of an SpmiStatVals structure.

Library

SPMI Library (libSpmi.a)

Syntax

#include sys/Spmidef.h
float SpmiGetValue(StatSet, StatVal)
struct SpmiStatSet *StatSet;
struct SpmiStatVals *StatVal;

Description

The SpmiGetValue subroutine performs the following steps:

1. Verifies that an SpmiStatVals structure exists in the set of statistics identified by the StatSet parameter.
2. Determines the format of the data field as being either SiFloat or SiLong and extracts the data value for further processing.
3. Determines the data value as being of either type SiQuantity or type SiCounter.
4. If the data value is of type SiQuantity, returns the val field of the SpmiStatVals structure.
5. If the data value is of type SiCounter, returns the value of the val_change field of the SpmiStatVals structure divided by the elapsed number of seconds since the previous time a data value was requested for this set of statistics.
This subroutine call should only be issued after an SpmiGetStatSet subroutine has been issued against the statset.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

**Parameters**

**StatSet**

Specifies a pointer to a valid structure of type SpmiStatSet as created by the SpmiCreateStatSet subroutine call.

**StatVal**

Specifies a pointer to a valid structure of type SpmiStatVals as created by the SpmiPathAddSetStat subroutine call or returned by the SpmiFirstVals or SpmiNextVals subroutine calls.

**Return Values**

The SpmiGetValue subroutine returns the decoded value if successful. If unsuccessful, the subroutine returns a negative value that has a numerical value of at least 1.1.

**Error Codes**

All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char SpmiErrmsg[];
- extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrmsg variable contains text, in English, explaining the cause of the error. See the List of SPMI Error Codes for more information.

**Files**

/usr/include/sys/Spmidef.h

Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

**Related Information**

For related information, see:

- "SpmiGetStatSet Subroutine" on page 261
- "SpmiCreateStatSet Subroutine" on page 242
- "SpmiPathAddSetStat Subroutine" on page 274
- Understanding SPMI Data Areas

**Spmilnit Subroutine**

**Purpose**

Initializes the SPMI for a local data consumer program.
Library
SPMI Library (libSpmi.a)

Syntax
#include sys/Spmidef.h
int SpmiInit (TimeOut)
int TimeOut;

Description
The SpmilInit subroutine initializes the SPMI. During SPMI initialization, a memory segment is allocated
and the application program obtains basic addressability to that segment. An application program must
issue the SpmilInit subroutine call before issuing any other subroutine calls to the SPMI.

Note: The SpmilInit subroutine is automatically issued by the SpmiDdsInit subroutine call. Successive
SpmilInit subroutine calls are ignored.

Note: If the calling program uses shared memory for other purposes, including memory mapping of files,
the SpmilInit subroutine call must be issued before access is established to other shared memory
areas.

The SPMI entry point called by the SpmilInit subroutine assigns a segment register to be used by the
SPMI subroutines (and the application program) for accessing common shared memory and establishes
the access mode to the common shared memory segment. After SPMI initialization, the SPMI subroutines
are able to access the common shared memory segment in read-only mode.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also
included in the Performance Toolbox for AIX licensed product.

Parameters
TimeOut

Specifies the number of seconds the SPMI waits for a Dynamic Data Supplier (DDS) program to
update its shared memory segment. If a DDS program does not update its shared memory
segment in the time specified, the SPMI assumes that the DDS program has terminated or
disconnected from shared memory and removes all contexts and statistics added by the DDS
program.

The SPMI saves the largest TimeOut value received from the programs that invoke the SPMI. The
TimeOut value must be zero or must be greater than or equal to 15 seconds and less than or
equal to 600 seconds. A value of zero overrides any other value from any other program that
invokes the SPMI and disables the checking for terminated DDS programs.

Return Values
The SpmilInit subroutine returns a value of 0 if successful. If unsuccessful, the subroutine returns a
nonzero value. If a nonzero value is returned, the application program should not attempt to issue
additional SPMI subroutine calls.

Error Codes
All SPMI subroutines use external variables to provide error information. To access these variables, an
application program must define the following external variables:
- extern char SpmiErrmsg[];
- extern int SpmiErrno;
If the subroutine returns without an error, the `SpmiErnno` variable is set to 0 and the `SpmiErrmsg` character array is empty. If an error is detected, the `SpmiErnno` variable returns an error code, as defined in the `sys/Spmidef.h` file, and the `SpmiErrmsg` variable contains text, in English, explaining the cause of the error. See the [List of SPMI Error Codes](#) for more information.

**Files**

/usr/include/sys/Spmidef.h

Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

**Related Information**

For related information, see:

- “SpmiDdsInit Subroutine” on page 245
- “SpmiExit Subroutine” on page 250

**SpmiInstantiate Subroutine**

**Purpose**

Explicitly instantiates the subcontexts of an instantiable context.

**Library**

SPMI Library (`libSpmi.a`)

**Syntax**

```c
#include sys/Spmidef.h
int SpmiInstantiate(CxHandle) 
SpmiCxHdl CxHandle;
```

**Description**

The `SpmiInstantiate` subroutine explicitly instantiates the subcontexts of an instantiable context. If the context is not instantiable, do not call the `SpmiInstantiate` subroutine.

An instantiation is done implicitly by the `SpmiPathGetCx` and `SpmiFirstCx` subroutine calls. Therefore, application programs usually do not need to instantiate explicitly.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

**Parameters**

**CxHandle**

Specifies a valid context handle `SpmiCxHdl` as obtained by another subroutine call.

**Return Values**

The `SpmiInstantiate` subroutine returns a value of 0 if successful. If the context is not instantiable, the subroutine returns a nonzero value.

**Error Codes**

All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:
extern char SpmiErrmsg[];
extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrmsg variable contains text, in English, explaining the cause of the error. See the List of SPMI Error Codes for more information.

Files
/usr/include/sys/Spmidef.h

Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

Related Information
For related information, see:
- "SpmiFirstCx Subroutine" on page 250
- "SpmiPathGetCx Subroutine" on page 276
- Understanding the SPMI Data Hierarchy

SpmiNextCx Subroutine

Purpose
Locates the next subcontext of a context.

Library
SPMI Library (libSpmi.a)

Syntax
#include sys/Spmidef.h
struct SpmiCxLink *SpmiNextCx(CxLink)

Description
The SpmiNextCx subroutine locates the next subcontext of a context, taking the context identified by the CxLink parameter as the current subcontext. The subroutine returns a NULL value if no further subcontexts are found.

The structure pointed to by the returned pointer contains an SpmiCxHdl handle to access the contents of the corresponding SpmiCx structure through the SpmiGetCx subroutine call.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

Parameters
CxLink
Specifies a pointer to a valid SpmiCxLink structure as obtained by a previous SpmiFirstCx subroutine.
Return Values
The SpmiNextCx subroutine returns a pointer to a structure of type SpmiCxLink if successful. If unsuccessful, the subroutine returns a NULL value.

Error Codes
All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char SpmiErrmsg[];
- extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrmsg variable contains text, in English, explaining the cause of the error. See the List of SPMI Error Codes for more information.

Files
/usr/include/sys/Spmidef.h
Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

Related Information
For related information, see:
- “SpmiFirstCx Subroutine” on page 250
- “SpmiGetCx Subroutine” on page 257
- Understanding SPMI Data Areas
- Understanding the SPMI Data Hierarchy

SpmiNextHot Subroutine

Purpose
Locates the next set of peer statistics SpmiHotVals belonging to an SpmiHotSet

Library
SPMI Library (libSpmi.a)

Syntax
#include sys/Spmidef.h
struct SpmiHotVals *SpmiNextHot(HotSet, HotVals)
struct SpmiHotSet *HotSet;
struct SpmiHotVals *HotVals;

Description
The SpmiNextHot subroutine locates the next SpmiHotVals structure belonging to an SpmiHotSet, taking the set of peer statistics identified by the HotVals parameter as the current one. The subroutine returns a NULL value if no further SpmiHotVals structures are found. The SpmiNextHot subroutine should only be executed after a successful call to the SpmiGetHotSet subroutine and (usually, but not necessarily) a call to the SpmiFirstHot subroutine and one or more subsequent calls to SpmiNextHot.

The subroutine allows the application programmer to position at the next set of peer statistics in preparation for using the SpmiNextHotItem subroutine call to traverse this peer set’s array of
elements. Use of this subroutine is only necessary if it is desired to skip over some structures in an SpmiHotSet. Under most circumstances, the SpmiNextHotItem will be the sole means of accessing all elements of the SpmiHotItems arrays of all peer sets belonging to an SpmiHotSet.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

**Parameters**

**HotSet**

Specifies a valid pointer to an SpmiHotSet structure as obtained by a previous SpmiCreateHotSet call.

**HotVals**

Specifies a pointer to an SpmiHotVals structure as returned by a previous SpmiFirstHot or SpmiNextHot subroutine call or as returned by an SpmiAddSetHot subroutine call.

**Return Values**

The SpmiNextHot subroutine returns a pointer to the next SpmiHotVals structure within the hotset. If no more SpmiHotVals structures are available, the subroutine returns a NULL value. A returned pointer may refer to a pseudo-hotvals structure as described the SpmiAddSetHot subroutine.

**Error Codes**

All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- extern char SpmiErrmsg[];
- extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrmsg variable contains text, in English, explaining the cause of the error. See the [List of SPMI Error Codes](#) for more information.

**Files**

/usr/include/sys/Spmidef.h

Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

**Related Information**

For more information, see:

- “SpmiFirstHot Subroutine” on page 251
- “SpmiGetHotSet Subroutine” on page 258
- “SpmiNextHotItem Subroutine”
- [Data Access Structures and Handles, HotSets](#)

**SpmiNextHotItem Subroutine**

**Purpose**

Locates and decodes the next SpmiHotItems element at the current position in an SpmiHotSet.
Library

SPMI Library (libSpmi.a)

Syntax

```c
#include sys/Spmidef.h
struct SpmiHotVals *SpmiNextHotItem(HotSet, HotVals, index, value, name)
struct SpmiHotSet *HotSet;
struct SpmiHotVals *HotVals;
int *Index;
float *value;
char **name;
```

Description

The `SpmiNextHotItem` subroutine locates the next `SpmiHotItems` structure belonging to an `SpmiHotSet`, taking the element identified by the `HotVals` and `index` parameters as the current one. The subroutine returns a NULL value if no further `SpmiHotItems` structures are found. The `SpmiNextHotItem` subroutine should only be executed after a successful call to the `SpmiGetHotSet` subroutine.

The `SpmiNextHotItem` subroutine is designed to be used for walking all `SpmiHotItems` elements returned by a call to the `SpmiGetHotSet` subroutine, visiting the `SpmiHotVals` structures one by one. By feeding the returned value and the updated integer pointed to by `index` back to the next call, this can be done in a tight loop. Successful calls to `SpmiNextHotItem` will decode each `SpmiHotItems` element and return the data value in `value` and the name of the peer context that owns the corresponding statistic in `name`.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

Parameters

**HotSet**

Specifies a valid pointer to an `SpmiHotSet` structure as obtained by a previous `SpmiCreateHotSet` on page 241 call.

**HotVals**

Specifies a pointer to an `SpmiHotVals` structure as returned by a previous `SpmiNextHotItem`, `SpmiFirstHot`, or `SpmiNextHot` subroutine call or as returned by an `SpmiAddSetHot` subroutine call. If this parameter is specified as NULL, the first `SpmiHotVals` structure of the `SpmiHotSet` is used and the `index` parameter is assumed to be set to zero, regardless of its actual value.

**index**

A pointer to an integer that contains the desired element number in the `SpmiHotItems` array of the `SpmiHotVals` structure specified by `HotVals`. A value of zero points to the first element. When the `SpmiNextHotItem` subroutine returns, the integer contain the index of the next `SpmiHotItems` element within the returned `SpmiHotVals` structure. If the last element of the array is decoded, the value in the integer will point beyond the end of the array, and the `SpmiHotVals` pointer returned will point to the peer set, which has now been completely decoded. By passing the returned `SpmiHotVals` pointer and the `index` parameter to the next call to `SpmiNextHotItem`, the subroutine will detect this and proceed to the first `SpmiHotItems` element of the next `SpmiHotVals` structure if one exists.

**value**

A pointer to a float variable. A successful call will return the decoded data value for the statistic. Before the value is returned, the `SpmiNextHotItem` function:
• Determines the format of the data field as being either SiFloat or SiLong and extracts the data value for further processing.

• Determines the data value as being either type SiQuantity or type SiCounter and performs one of the actions listed here:
  – If the data value is of type SiQuantity, the subroutine returns the val field of the SpmiHotItems structure.
  – If the data value is of type SiCounter, the subroutine returns the value of the val_change field of the SpmiHotItems structure divided by the elapsed number of seconds since the previous time a data value was requested for this set of statistics.

name
A pointer to a character pointer. A successful call will return a pointer to the name of the peer context for which the data value was read.

Return Values
The SpmiNextHotItem subroutine returns a pointer to the current SpmiHotVals structure within the hotset. If no more SpmiHotVals structures are available, the subroutine returns a NULL value. The structure returned contains the data, such as threshold, which may be relevant for presentation of the results of an SpmiGetHotSet subroutine call to end-users. A returned pointer may refer to a pseudo-hotvals structure as described in the SpmiAddSetHot subroutine.

Error Codes
All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:
• extern char SpmiErrmsg[];
• extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrmsg variable contains text, in English, explaining the cause of the error. See the List of SPMI Error Codes for more information.

Files
/usr/include/sys/Spmidef.h Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

Related Information
For more information, see:
• “SpmiFirstHot Subroutine” on page 251
• “SpmiNextHot Subroutine” on page 267
• “SpmiGetHotSet Subroutine” on page 258
• Data Access Structures and Handles, HotSets

SpmiNextStat Subroutine

Purpose
Locates the next statistic belonging to a context.
Library
SPMI Library (libSpmi.a)

Syntax
#include sys/Spmidef.h
struct SpmiStatLink *SpmiNextStat(StatLink)
struct SpmiStatLink *StatLink;

Description
The SpmiNextStat subroutine locates the next statistic belonging to a context, taking the statistic identified by the StatLink parameter as the current statistic. The subroutine returns a NULL value if no further statistics are found.

The structure pointed to by the returned pointer contains an SpmiStatHdl handle to access the contents of the corresponding SpmiStat structure through the “SpmiGetStat Subroutine” on page 259 call.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

Parameters
StatLink
   Specifies a valid pointer to a SpmiStatLink structure as obtained by a previous SpmiFirstStat Subroutine” on page 252 call.

Return Values
The SpmiNextStat subroutine returns a pointer to a structure of type SpmiStatLink if successful. If unsuccessful, the subroutine returns a NULL value.

Error Codes
All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:
   • extern char SpmiErrmsg[];
   • extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrmsg variable contains text, in English, explaining the cause of the error. See the List of SPMI Error Codes for more information.

Files
/usr/include/sys/Spmidef.h
   Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

Related Information
For related information, see:
   • “SpmiFirstStat Subroutine” on page 252
   • “SpmiGetStat Subroutine” on page 259
   • Understanding SPMI Data Areas
Understanding the SPMI Data Hierarchy

SpmiNextVals Subroutine

Purpose
Returns a pointer to the next SpmiStatVals structure in a set of statistics.

Library
SPMI Library (libSpmi.a)

Syntax
#include sys/Spmidef.h
struct SpmiStatVals *SpmiNextVals(StatSet, StatVal)
struct SpmiStatSet *StatSet;
struct SpmiStatVals *StatVal;

Description
The SpmiNextVals subroutine returns a pointer to the next SpmiStatVals structure in a set of statistics, taking the structure identified by the StatVal parameter as the current structure. The SpmiStatVals structures are accessed in reverse order so the statistic added before the current one is returned. This subroutine call should only be issued after an SpmiGetStatSet subroutine has been issued against the statset.

Parameters
StatSet
Specifies a pointer to a valid structure of type SpmiStatSet as created by the SpmiCreateStatSet Subroutine on page 242 call.

StatVal
Specifies a pointer to a valid structure of type SpmiStatVals as created by the SpmiPathAddSetStat Subroutine on page 274 subroutine call or returned by a previous SpmiFirstVals Subroutine on page 254 or SpmiNextVals subroutine call.

Return Values
The SpmiNextVals subroutine returns a pointer to a SpmiStatVals structure if successful. If unsuccessful, the subroutine returns a NULL value.

SpmiNextValue Subroutine

Purpose
Returns either the first SpmiStatVals structure in a set of statistics or the next SpmiStatVals structure in a set of statistics and a decoded value based on the type of data value extracted from the data field of an SpmiStatVals structure.

Library
SPMI Library (libSpmi.a)

Syntax
#include sys/Spmidef.h
Description

Instead of issuing subroutine calls to “SpmiFirstVals Subroutine” on page 254 / “SpmiNextVals Subroutine” on page 272 (to get the first or next SpmiStatVals structure) followed by calls to SpmiGetValue (to get the decoded value from the SpmiStatVals structure), the SpmiNextValue subroutine returns both in one call. This subroutine call returns a pointer to the first SpmiStatVals structure belonging to the StatSet parameter if the StatVal parameter is NULL. If the StatVal parameter is not NULL, the next SpmiStatVals structure is returned, taking the structure identified by the StatVal parameter as the current structure. The data value corresponding to the returned SpmiStatVals structure is decoded and returned in the field pointed to by the value argument. In decoding the data value, the subroutine does the following:

- Determines the format of the data field as being either SiFloat or SiLong and extracts the data value for further processing.
- Determines the data value as being either type SiQuantity or type SiCounter and performs one of the actions listed here:
  - If the data value is of type SiQuantity, the subroutine returns the val field of the SpmiStatVals structure.
  - If the data value is of type SiCounter, the subroutine returns the value of the val_change field of the SpmiStatVals structure divided by the elapsed number of seconds since the previous time a data value was requested for this set of statistics.

Note: This subroutine call should only be issued after an “SpmiGetStatSet Subroutine” on page 261 has been issued against the statset.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

Parameters

StatSet

Specifies a pointer to a valid structure of type SpmiStatSet as created by the “SpmiCreateStatSet Subroutine” on page 242 call.

StatVal

Specifies either a NULL pointer or a pointer to a valid structure of type SpmiStatVals as created by the “SpmiPathAddSetStat Subroutine” on page 274 call or returned by a previous SpmiNextValue subroutine call. If StatVal is NULL, then the first SpmiStatVals pointer belonging to the set of statistics pointed to by StatSet is returned.

value

A pointer used to return a decoded value based on the type of data value extracted from the data field of the returned SpmiStatVals structure.

Return Value

The SpmiNextValue subroutine returns a pointer to a SpmiStatVals structure if successful. If unsuccessful, the subroutine returns a NULL value.

If the StatVal parameter is:

NULL

The first SpmiStatVals structure belonging to the StatSet parameter is returned.
The next `SpmiStatVals` structure after the structure identified by the `StatVal` parameter is returned and the value parameter is used to return a decoded value based on the type of data value extracted from the data field of the returned `SpmiStatVals` structure.

### Error Codes

All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- `extern char SpmiErrmsg[];
- `extern int SpmiErrno;

If the subroutine returns without an error, the `SpmiErrno` variable is set to 0 and the `SpmiErrmsg` character array is empty. If an error is detected, the `SpmiErrno` variable returns an error code, as defined in the `sys/Spmidef.h` file, and the `SpmiErrmsg` variable contains text, in English, explaining the cause of the error. See the [List of SPMI Error Codes](#) for more information.

### Programming Notes

The `SpmiNextValue` subroutine maintains internal state information so that retrieval of the next data value from a statset can be done without traversing linked lists of data structures. The stats information is kept separate for each process, but is shared by all threads of a process.

If the subroutine is accessed from multiple threads, the state information is useless and the performance advantage is lost. The same is true if the program is simultaneously accessing two or more statsets. To benefit from the performance advantage of the `SpmiNextValue` subroutine, a program should retrieve all values in order from one stat set before retrieving values from the next statset.

The implementation of the subroutine allows a program to retrieve data values beginning at any point in the statset if the `SpmiStatVals` pointer is known. Doing so will cause a linked list traversal. If subsequent invocations of `SpmiNextValue` uses the value returned from the first and following invocation as their second argument, the traversal of the link list can be avoided.

It should be noted that the value returned by a successful `SpmiNextValue` invocation is always the pointer to the `SpmiStatVals` structure whose data value is decoded and returned in the value argument.

### Files

`/usr/include/sys/Spmidef.h` Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

### Related Information

For related information, see:

- “`SpmiGetStatSet Subroutine`” on page 261
- “`SpmiCreateStatSet Subroutine`” on page 242
- “`SpmiPathAddSetStat Subroutine.`”
- `Data Access Structures and Handles, StatSets`

---

**SpmiPathAddSetStat Subroutine**

### Purpose

Adds a statistics value to a set of statistics.
Library
SPMI Library (libSpmi.a)

Syntax
#include <sys/Spmidef.h>
struct SpmiStatVals *SpmiPathAddSetStat(StatSet, StatName, Parent);
struct SpmiStatSet *StatSet;
char *StatName;
SpmiCxHdl Parent;

Description
The SpmiPathAddSetStat subroutine adds a statistics value to a set of statistics. The SpmiStatSet structure that provides the anchor point to the set must exist before the SpmiPathAddSetStat subroutine call can succeed.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

Parameters
StatSet
Specifies a pointer to a valid structure of type SpmiStatSet as created by the SpmiCreateStatSet Subroutine call.

StatName
Specifies the name of the statistic within the context identified by the Parent parameter. If the Parent parameter is NULL, you must specify the fully qualified path name of the statistic in the StatName parameter.

Parent
Specifies either a valid SpmiCxHdl handle as obtained by another subroutine call or a NULL value.

Return Values
The SpmiPathAddSetStat subroutine returns a pointer to a structure of type SpmiStatVals if successful. If unsuccessful, the subroutine returns a NULL value.

Error Codes
All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:
• extern char SpmiErrmsg[];
• extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrmsg variable contains text, in English, explaining the cause of the error. See the List of SPMI Error Codes for more information.

Files
/usr/include/sys/Spmidef.h
Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.
Related Information
For related information, see:

- “SpmiGetStatSet Subroutine” on page 261
- “SpmiCreateStatSet Subroutine” on page 242
- “SpmiDelSetStat Subroutine” on page 248
- “SpmiFreeStatSet Subroutine” on page 256
- Data Access Structures and Handles, StatSets

SpmiPathGetCx Subroutine

Purpose
Returns a handle to use when referencing a context.

Library
SPMI Library (libSpmi.a)

Syntax
#include sys/Spmidef.h
SpmiCxHdl SpmiPathGetCx(CxPath, Parent)
char *CxPath;
SpmiCxHdl Parent;

Description
The SpmiPathGetCx subroutine searches the context hierarchy for a given path name of a context and returns a handle to use when subsequently referencing the context.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.

Parameters
CxPath
Specifications the path name of the context to find. If you specify the fully qualified path name in the CxPath parameter, you must set the Parent parameter to NULL. If the path name is not qualified or is only partly qualified (that is, if it does not include the names of all contexts higher in the data hierarchy), the SpmiPathGetCx subroutine begins searching the hierarchy at the context identified by the Parent parameter. If the CxPath parameter is either NULL or an empty string, the subroutine returns a handle identifying the Top context.

Parent
Specifies the anchor context that fully qualifies the CxPath parameter. If you specify a fully qualified path name in the CxPath parameter, you must set the Parent parameter to NULL.

Return Values
The SpmiPathGetCx subroutine returns a handle to a context if successful. If unsuccessful, the subroutine returns a NULL value.
**Error Codes**

All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

- `extern char SpmiErrmsg[];
- `extern int SpmiErrno;

If the subroutine returns without an error, the `SpmiErrno` variable is set to 0 and the `SpmiErrmsg` character array is empty. If an error is detected, the `SpmiErrno` variable returns an error code, as defined in the `sys/Spmidef.h` file, and the `SpmiErrmsg` variable contains text, in English, explaining the cause of the error. See the [List of SPMI Error Codes](#) for more information.

**Files**

`/usr/include/sys/Spmidef.h`  
Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

**Related Information**

For related information, see:

- [Understanding SPMI Data Areas](#)
- [Understanding the SPMI Data Hierarchy](#)

---

**SpmiStatGetPath Subroutine**

**Purpose**

Returns the full path name of a statistic.

**Library**

SPMI Library (libSpmi.a)

**Syntax**

```c
#include sys/Spmidef.h>

char *miStatGetPath(Parent, StatHandle, MaxLevels)
SpmiCxHdlSp Parent;
SpmiStatHdl StatHandle;
int MaxLevels;
```

**Description**

The `SpmiStatGetPath` subroutine returns the full path name of a statistic, given a parent context `SpmiCxHdl` handle and a statistics `SpmiStatHdl` handle. The `MaxLevels` parameter can limit the number of levels in the hierarchy that must be searched to generate the path name of the statistic.

The memory area pointed to by the returned pointer is freed when the `SpmiStatGetPath` subroutine call is repeated. For each invocation of the subroutine, a new memory area is allocated and its address returned. If the calling program needs the returned character string after issuing the `SpmiStatGetPath` subroutine call, the program must copy the returned string to locally allocated memory before reissuing the subroutine call.

This subroutine is part of the server option of the Performance Aide for AIX licensed product and is also included in the Performance Toolbox for AIX licensed product.
Parameters

Parent

Specifies a valid SpmiCxHdl handle as obtained by another subroutine call.

StatHandle

Specifies a valid SpmiStatHdl handle as obtained by another subroutine call. This handle must point to a statistic belonging to the context identified by the Parent parameter.

MaxLevels

Limits the number of levels in the hierarchy that must be searched to generate the path name. If this parameter is set to 0, no limit is imposed.

Return Values

If successful, the SpmiStatGetPath subroutine returns a pointer to a character array containing the full path name of the statistic. If unsuccessful, the subroutine returns a NULL value.

Error Codes

All SPMI subroutines use external variables to provide error information. To access these variables, an application program must define the following external variables:

• extern char SpmiErrmsg[];
• extern int SpmiErrno;

If the subroutine returns without an error, the SpmiErrno variable is set to 0 and the SpmiErrmsg character array is empty. If an error is detected, the SpmiErrno variable returns an error code, as defined in the sys/Spmidef.h file, and the SpmiErrmsg variable contains text, in English, explaining the cause of the error. See the List of SPMI Error Codes for more information.

Files

/usr/include/sys/Spmidef.h

Declares the subroutines, data structures, handles, and macros that an application program can use to access the SPMI.

Related Information

For related information, see:

• Understanding SPMI Data Areas
• Understanding the SPMI Data Hierarchy

sqrt, sqrtf, or sqrtl Subroutine

Purpose

Computes the square root.

Syntax

```
#include <math.h>
double sqrt ( x )
double x;
float sqrtf ( x )
float x;
```
long double sqrtl (x)
long double x;

Description
The sqrt, sqrtf, and sqrtl subroutines compute the square root of the x parameter.

An application wishing to check for error situations should set the errno global variable to zero and call feclearexcept(FE_ALL_EXCEPT) before calling these subroutines. Upon return, if errno is nonzero or fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is nonzero, an error has occurred.

Parameters
x Specifies some double-precision floating-point value.

Return Values
Upon successful completion, the sqrtf subroutine returns the square root of x.

For finite values of x < -0, a domain error occurs, and a NaN is returned.

If x is NaN, a NaN is returned.

If x is ±0 or +Inf, x is returned.

If x is -Inf, a domain error shall occur, and a NaN is returned.

Error Codes
When using libm.a (-lm):

For the sqrt subroutine, if the value of x is negative, a NaNQ is returned and the errno global variable is set to a EDOM value.

When using libmsaa.a (-lmsaa):

If the value of x is negative, a 0 is returned and the errno global variable is set to a EDOM value. A message indicating a DOMAIN error is printed on the standard error output.

These error-handling procedures may be changed with the matherr subroutine when using the libmsaa.a (-lmsaa) library.

Related Information
The exp, expm1, log, log10, log1p, or pow subroutine.

Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
src_err_msg Subroutine

Purpose
Retrieves a System Resource Controller (SRC) error message.

Library
System Resource Controller Library (linsrc.a)

Syntax
```c
int src_err_msg (errno, ErrorText)
int errno;
char **ErrorText;
```

Description
The src_err_msg subroutine retrieves a System Resource Controller (SRC) error message.

Parameters
- **errno** Specifies the SRC error code.
- **ErrorText** Points to a character pointer to place the SRC error message.

Return Values
Upon successful completion, the src_err_msg subroutine returns a value of 0. Otherwise, a value of -1 is returned. No error message is returned.

Related Information
The addsys subroutine, chsys subroutine, delsys subroutine, defsys subroutine, getsys subroutine, getssys subroutine, srcbuf subroutine, srcbuf Subroutine on page 281, srcrqs subroutine, srcrqs Subroutine on page 281, srcsbuf subroutine, srcsbuf Subroutine on page 284, srcsrpy subroutine, srcsrpy Subroutine on page 290, srcsrqt subroutine, srcsrqt Subroutine on page 293, srcstat subroutine, srcstat Subroutine on page 299, srcstathdr subroutine, srcstathdr Subroutine on page 304, srcsrqt Subroutine on page 305, srcstop subroutine, srcstop Subroutine on page 306, srcstrt subroutine, srcstrt Subroutine on page 308 subroutine.

List of SRC Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
src_err_msg_r Subroutine

Purpose
Gets the System Resource Controller (SRC) error message corresponding to the specified SRC error code.

Library
System Resource Controller (libsrc.a)

Syntax
#include <spc.h>

int src_err_msg_r (srcerrno, ErrorText)

int srcerrno;
char ** ErrorText;

Description
The src_err_msg_r subroutine returns the message corresponding to the input srcerrno value in a caller-supplied buffer. This subroutine is threadsafe and reentrant.

Parameters
srcerrno Specifies the SRC error code.
ErrorText Pointer to a variable containing the address of a caller-supplied buffer where the message will be returned. If the length of the message is unknown, the maximum message length can be used when allocating the buffer. The maximum message length is SRC_BUF_MAX in /usr/include/spc.h (2048 bytes).

Return Values
Upon successful completion, the src_err_msg_r subroutine returns a value of 0. Otherwise, no error message is returned and the subroutine returns a value of -1.

Related Information
The srcsbuf_r ("srcsbuf_r Subroutine" on page 287), srcsrqt_r ("srcsrqt_r Subroutine" on page 296), srcrrqs_r ("srcrrqs_r Subroutine" on page 283), srcstat_r ("srcstat_r Subroutine" on page 302), and srcstattxt_r ("srcstattxt_r Subroutine" on page 305) subroutines.

List of SRC Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Programming Subsystem Communication with the SRC in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

System Resource Controller (SRC) Overview for Programmers in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

srcrrqs Subroutine

Purpose
Gets subsystem reply information from the System Resource Controller (SRC) request received.
Library
System Resource Controller Library (libsrc.a)

Syntax
#include <spc.h>

struct srchdr *srcrrqs (Packet)
char *Packet;

Description
The srcrrqs subroutine saves the srchdr information contained in the packet the subsystem received from the System Resource Controller (SRC). The srchdr structure is defined in the spc.h file. This routine must be called by the subsystem to complete the reception process of any packet received from the SRC. The subsystem requires this information to reply to any request that the subsystem receives from the SRC.

Note: The saved srchdr information is overwritten each time this subroutine is called.

Parameters
Packet Points to the SRC request packet received by the subsystem. If the subsystem received the packet on a message queue, the Packet parameter must point past the message type of the packet to the start of the request information. If the subsystem received the information on a socket, the Packet parameter points to the start of the packet received on the socket.

Return Values
The srcrrqs subroutine returns a pointer to the static srchdr structure, which contains the return address for the subsystem response.

Examples
The following will obtain the subsystem reply information:

```c
int rc;
struct sockaddr addr;
int addrsz;
struct srcreq packet;

/* wait to receive packet from SRC daemon */
rc=recvfrom(0, &packet, sizeof(packet), 0, &addr, &addrsz);
/* grab the reply information from the SRC packet */
if (rc>0)
    srchdr=srcrrqs (&packet);
```

Files
/dev/SRC Specifies the AF_UNIX socket file.
/dev/.SRC-unix Specifies the location for temporary socket files.

Related Information
The srcsbuf subroutine, srcsreq subroutine, srcsreqf subroutine, srcstat subroutine, srcstatf subroutine, srcstattx subroutine, srcstattxn subroutine, srcstop subroutine, srcstrt subroutine.
### srcrrqs_r Subroutine

#### Purpose
Copies the System Resource Controller (SRC) request header to the specified buffer. The SRC request header contains the return address where the caller sends responses for this request.

#### Library
System Resource Controller (libsrc.a)

#### Syntax
```c
#include <spc.h>

struct srchdr *srcrrqs_r (Packet, SRChdr)
char * Packet;
struct srchdr * SRChdr;
```

#### Description
The `srcrrqs_r` subroutine saves the SRC request header (srchdr) information contained in the packet the subsystem received from the Source Resource Controller. The `srchdr` structure is defined in the `spc.h` file. This routine must be called by the subsystem to complete the reception process of any packet received from the SRC. The subsystem requires this information to reply to any request that the subsystem receives from the SRC.

This subroutine is threadsafe and reentrant.

#### Parameters
- **Packet** Points to the SRC request packet received by the subsystem. If the subsystem received the packet on a message queue, the `Packet` parameter must point past the message type of the packet to the start of the request information. If the subsystem received the information on a socket, the `Packet` parameter points to the start of the packet received on the socket.
- **SRChdr** Points to a caller-supplied buffer. The `srcrrqs_r` subroutine copies the request header to this buffer.

#### Examples
The following will obtain the subsystem reply information:
```c
int rc;
struct sockaddr addr;
int addrsz;
struct srcreq packet;
struct srchdr *header;
struct srchdr *rtn_addr;

/*wait to receive packet from SRC daemon */
rc=recvfrom(0, &packet, sizeof(packet), 0, &addr, &addrsz;
```
/* grab the reply information from the SRC packet */
if (rc>0)
{
    header = (struct srchdr *)malloc(sizeof(struct srchdr));
    rtn_addr = srcrrqs_r(&packet,header);
    if (rtn_addr == NULL)
    {
        /* handle error */
    }
}

Return Values
Upon successful completion, the srcrrq_r subroutine returns the address of the caller-supplied buffer.

Error Codes
If either of the input addresses is NULL, the srcrrqs_r subroutine fails and returns a value of NULL.

SRC_PARM One of the input addresses is NULL.

Related Information
The src_err_msg_r ("src_err_msg_r Subroutine" on page 281), srcbuf_r ("srcbuf_r Subroutine" on page 287), srcsrqt_r ("srcsrqt_r Subroutine" on page 296), srcstat_r ("srcstat_r Subroutine" on page 302), and srcstattxt_r ("srcstattxt_r Subroutine" on page 305) subroutines.

srcbuf Subroutine

Purpose
Gets status for a subserver or a subsystem and returns status text to be printed.

Library
System Resource Controller Library (libsrc.a)

Syntax
#include <spc.h>

int srcbuf(Host, Type, SubsystemName, SubserverObject, SubsystemPID, StatusType, StatusFrom, StatusText, Continued)

  char * Host, * SubsystemName;
  char * SubserverObject, **>StatusText;
  short Type, StatusType;
  int SubsystemPID, StatusFrom, * Continued;

Description
The srcbuf subroutine gets the status of a subserver or subsystem and returns printable text for the status in the address pointed to by the StatusText parameter.

When the StatusType parameter is SHORTSTAT and the Type parameter is SUBSYSTEM, the srcstat subroutine is called to get the status of one or more subsystems. When the StatusType parameter is
LONGSTAT and the Type parameter is SUBSYSTEM, the srcsqt subroutine is called to get the long status of one subsystem. When the Type parameter is not SUBSYSTEM, the srcsqt subroutine is called to get the long or short status of a subserver.

Parameters

Host Specifies the foreign host on which this status action is requested. If the host is null, the status request is sent to the System Resource Controller (SRC) on the local host. The local user must be running as "root". The remote system must be configured to accept remote System Resource Controller requests. That is, the srcmstr daemon (see /etc/inittab) must be started with the -r flag and the /etc/hosts.equiv or .rhosts file must be configured to allow remote requests.

Type Specifies whether the status request applies to the subsystem or subserver. If the Type parameter is set to SUBSYSTEM, the status request is for a subsystem. If not, the status request is for a subserver and the Type parameter is a subserver code point.

SubsystemName Specifies the name of the subsystem on which to get status. To get the status of all subsystems, use the SRCALLSUBSYS constant. To get the status of a group of subsystems, the SubsystemName parameter must start with the SRCGROUP constant, followed by the name of the group for which you want status appended. If you specify a null SubsystemName parameter, you must specify a SubsystemPID parameter.

SubserverObject Specifies a subserver object. The SubserverObject parameter modifies the Type parameter. The SubserverObject parameter is ignored if the Type parameter is set to SUBSYSTEM. The use of the SubserverObject parameter is determined by the subsystem and the caller. This parameter will be placed in the objname field of the subreq structure that is passed to the subsystem.

SubsystemPID Specifies the process ID of the subsystem on which to get status, as returned by the srcstrt subroutine. You must specify the SubsystemPID parameter if multiple instances of the subsystem are active and you request a long subsystem status or subserver status. If you specify a null SubsystemPID parameter, you must specify a SubsystemName parameter.

StatusType Specifies LONGSTAT for long status or SHORTSTAT for short status.

StatusFrom Specifies whether status errors and messages are to be printed to standard output or just returned to the caller. When the StatusFrom parameter is SSHELL, the errors are printed to standard output.

StatusText Allocates memory for the printable text and sets the StatusText parameter to point to this memory. After it prints the text, the calling process must free the memory allocated for this buffer.

Continued Specifies whether this call to the srcsbuf subroutine is a continuation of a status request. If the Continued parameter is set to NEWREQUEST, a request for status is sent and the srcsbuf subroutine then waits for another. On return, the srcsbuf subroutine is updated to the new continuation indicator from the reply packet and the Continued parameter is set to END or STATCONTINUED by the subsystem. If the Continued parameter is set to something other than END, this field must remain equal to that value; otherwise, this function will not be able to receive any more packets for the original status request. The calling process should not set the value of the Continued parameter to a value other than NEWREQUEST. The Continued parameter should not be changed while more responses are expected.

Return Values

If the srcsbuf subroutine succeeds, it returns the size (in bytes) of printable text pointed to by the StatusText parameter.

Error Codes

The srcsbuf subroutine fails if one or more of the following are true:

SRC_BADSOCK The request could not be passed to the subsystem because of some socket failure.
The subsystem uses signals. The request cannot complete.
The SRC daemon is not active.
The local host is not in the remote /etc/hosts.equiv file.
On the remote host, the local host is not known.
The user is not root or group system.
An SRC component could not allocate the memory it needs.
The Continued parameter was not set to NEWREQUEST, and no
continuation is currently active.
The request timed out waiting for a response.
The subsystem is not active.
There is a problem with SRC socket communications.
The request was not passed to the subsystem. The subsystem is
stopping.
The SRC port is not defined in the /etc/services file.
The foreign host is not known.
There are multiple instances of the subsystem active.

Examples

1. To get the status of a subsystem, enter:

   char *status;
   int continued=NEWREQUEST;
   int rc;

   do {
      rc=srcsbuf("MaryC", SUBSYSTEM, "srctest", ",", 0,
                   SHORTSTAT, SSHAEL, &status, continued);
      if (status!=0) {
         printf(status);
         free(status);
         status=0;
      }
   } while (rc>0);
   
   This gets short status of the srctest subsystem on the MaryC machine and prints the formatted status to standard output.

2. To get the status of a subserver, enter:

   char *status;
   int continued=NEWREQUEST;
   int rc;

   do {
      rc=srcsbuf("", 12345, "srctest", ",", 0,
                   LONGSTAT, SSHAEL, &status, continued);
      if (status!=0) {
         printf(status);
         free(status);
         status=0;
      }
   } while (rc>0);
   
   This gets long status for a specific subserver belonging to subsystem srctest. The subserver is the one having code point 12345. This request is processed on the local machine. The formatted status is printed to standard output.

Files

/etc/services

Defines sockets and protocols used for Internet services.
Specifies the AF_UNIX socket file.
Specifies the location for temporary socket files.

Related Information
The srcrrqs ("srcrrqs Subroutine" on page 281) subroutine, srcrpy ("srcrpy Subroutine" on page 290) subroutine, srcsrq ("srcsrq Subroutine" on page 293) subroutine, srcstat ("srcstat Subroutine" on page 299) subroutine, srcstathdr ("srcstathdr Subroutine" on page 304) subroutine, srcstattxt ("srcstattxt Subroutine" on page 305) subroutine, srcstop ("srcstop Subroutine" on page 306) subroutine, srcsttr ("srcsttr Subroutine" on page 308) subroutine.

List of SRC Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Programming Subsystem Communication with the SRC in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

System Resource Controller (SRC) Overview for Programmers in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

srcsbuf_r Subroutine

Purpose
Gets status for a subserver or a subsystem and returns status text to be printed.

Library
System Resource Controller Library (libsrc.a)

Syntax
#include <spc.h>

int srcsbuf_r(Host, Type, SubsystemName, SubserverObject, SubsystemPID, StatusType, StatusFrom, StatusText, Continued, SRCHandle)

char * Host, * SubsystemName;
char * SubserverObject, ** StatusText;
short Type, StatusType;
pid_t SubsystemPID;
int StatusFrom, * Continued;
char ** SRCHandle;

Description
The srcsbuf_r subroutine gets the status of a subserver or subsystem and returns printable text for the status in the address pointed to by the StatusText parameter. The srcsbuf_r subroutine supports all the functions of the srcbuf subroutine except the StatusFrom parameter.

When the StatusType parameter is SHORTSTAT and the Type parameter is SUBSYSTEM, the srcsstat_r subroutine is called to get the status of one or more subsystems. When the StatusType parameter is LONGSTAT and the Type parameter is SUBSYSTEM, the srcsqt_r subroutine is called to get the long status of one subsystem. When the Type parameter is not SUBSYSTEM, the srcsqt_r subroutine is called to get the long or short status of a subserver.

This routine is threadsafe and reentrant.
Parameters

**Host**
Specifies the foreign host on which this status action is requested. If the host is null, the status request is sent to the System Resource Controller (SRC) on the local host.

**Type**
Specifies whether the status request applies to the subsystem or subserver. If the Type parameter is set to `SUBSYSTEM`, the status request is for a subsystem. If not, the status request is for a subserver and the Type parameter is a subserver code point.

**SubsystemName**
Specifies the name of the subsystem on which to get status. To get the status of all subsystems, use the `SRCALLSUBSYS` constant. To get the status of a group of subsystems, the SubsystemName parameter must start with the `SRCGROUP` constant, followed by the name of the group for which you want status appended. If you specify a null SubsystemName parameter, you must specify a SubsystemPID parameter.

**SubserverObject**
Specifies a subserver object. The SubserverObject parameter modifies the Type parameter. The SubserverObject parameter is ignored if the Type parameter is set to `SUBSYSTEM`. The use of the SubserverObject parameter is determined by the subsystem and the caller. This parameter will be placed in the objname field of the subreq structure that is passed to the subsystem.

**SubsystemPID**
Specifies the process ID of the subsystem on which to get status, as returned by the srcstrt subroutine. You must specify the SubsystemPID parameter if multiple instances of the subsystem are active and you request a long subsystem status or subserver status. If you specify a null SubsystemPID parameter, you must specify a SubsystemName parameter.

**StatusType**
Specifies `LONGSTAT` for long status or `SHORTSTAT` for short status.

**StatusFrom**
Specifies whether status errors and messages are to be printed to standard output or just returned to the caller. When the StatusFrom parameter is `SSHELL`, the errors are printed to standard output. The `SSHELL` value is not recommended in a multithreaded environment since error messages to standard output may be interleaved in an unexpected manner.

**StatusText**
Allocates memory for the printable text and sets the StatusText parameter to point to this memory. After it prints the text, the calling process must free the memory allocated for this buffer.

**Continued**
Specifies whether this call to the srcsbuf_r subroutine is a continuation of a status request. If the Continued parameter is set to `NEWREQUEST`, a request for status is sent and the srcsbuf_r subroutine then waits for a reply. On return from the srcsbuf_r subroutine, the Continued parameter is updated to the new continuation indicator from the reply packet. The continuation indicator in the reply packet will be set to `END` or `STATCONTINUED` by the subsystem. If the Continued parameter is set to something other than `END`, the caller should not change that value; otherwise, this function will not be able to receive any more packets for the original status request. The calling process should not set the value of the Continued parameter to a value other than `NEWREQUEST`. In normal processing, the Continued parameter should not be changed while more responses are expected. The caller must continue to call the srcsbuf_r subroutine until END is received. As an alternative, call the srcsbuf_r subroutine with `Continued=SRC_CLOSE` to discard the remaining data, close the socket, and free the internal buffers.

**SRCHandle**
Identifies a request and its associated responses. Set to NULL by the caller for a `NEWREQUEST`. The srcsbuf_r subroutine saves a value in SRCHandle to allow srcsbuf_r continuation calls to use the same socket and internal buffers. The SRCHandle parameter should not be changed by the caller except for `NEWREQUESTs`.

Return Values
If the srcsbuf_r subroutine succeeds, it returns the size (in bytes) of printable text pointed to by the StatusText parameter.
Error Codes

The srcsbuf_r subroutine fails and returns the corresponding error code if one of the following error conditions is detected:

**SRC_BADSOCK**

The request could not be passed to the subsystem because of some socket failure.

**SRC_CONT**

The subsystem uses signals. The request cannot complete.

**SRC_DMNA**

The SRC daemon is not active.

**SRC_INET_AUTHORIZED_HOST**

The local host is not in the remote `/etc/hosts.equiv` file.

**SRC_INET_INVALID_HOST**

On the remote host, the local host is not known.

**SRC_INVALID_USER**

The user is not root or group system.

**SRC_MMRY**

An SRC component could not allocate the memory it needs.

**SRC_NOCONTINUE**

The `Continued` parameter was not set to NEWREQUEST, and no continuation is currently active.

**SRC_NORPLY**

The request timed out waiting for a response.

**SRC_NSVR**

The subsystem is not active.

**SRC.SOCK**

There is a problem with SRC socket communications.

**SRC_STPG**

The request was not passed to the subsystem. The subsystem is stopping.

**SRC_UDP**

The SRC port is not defined in the `/etc/services` file.

**SRC_UHOST**

The foreign host is not known.

**SRC_WICH**

There are multiple instances of the subsystem active.

Examples

1. To get the status of a subsystem, enter:

   ```c
   char *status;
   int continued=NEWREQUEST;
   int rc;
   char *handle
   
do {
      rc=srcsbuf_r("MaryC", SUBSYSTEM, "srctest", ",", 0,
         SHORTSTAT, SDAEMON, &status, continued, &handle);
      if (status!=0)
      {
         printf(status);
         free(status);
         status=0;
      }
   } while (rc>0);
   if (rc<0)
   {
      ...handle error from srcsbuf_r...
   }
   
This gets short status of the srctest subsystem on the MaryC machine and prints the formatted status to standard output.

   **Caution:** In a multithreaded environment, the caller must manage the sharing of standard output between threads. Set the `StatusFrom` parameter to SDAEMON to prevent unexpected error messages from being printed to standard output.

2. To get the status of a subserver, enter:

   ```c
   char *status;
   int continued=NEWREQUEST;
   int rc;
   char *handle
   
do {
      rc=srcsbuf_r("", "", "srctest", ",", 0,
   ```
LONGSTAT, SDAEMON, &status, continued, &handle);
if (status!=0)
{
    printf(status);
    free(status);
    status=0;
}
} while (rc>0);
if (rc<0)
{
    ...handle error from srcbuf_r...
}

This gets long status for a specific subserver belonging to subsystem srctest. The subserver is the one having code point 12345. This request is processed on the local machine. The formatted status is printed to standard output.

**CAUTION:**

In a multithreaded environment, the caller must manage the sharing of standard output between threads. Set the `StatusFrom` parameter to SDAEMON to prevent unexpected error messages from being printed to standard output.

**Related Information**

The `src_err_msg_r` subroutine, `srcsrqt_r` subroutine, `srcrrqs_r` subroutine, `srcstat_r` subroutine, `srcstattxt_r` subroutine.

**List of SRC Subroutines** in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

**Programming Subsystem Communication with the SRC** in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

**System Resource Controller (SRC) Overview for Programmers** in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

**srcsrpy Subroutine**

**Purpose**

Sends a reply to a request from the System Resource Controller (SRC) back to the client process.

**Library**

System Resource Controller Library (libsrc.a)

**Syntax**

```
#include <spc.h>

int srcsrpy (SRChdr, PPacket, PPacketSize, Continued)
struct srchdr *SRChdr;
char *PPacket;
int PPacketSize;
ushort Continued;
```

**Description**

The `srcsrpy` subroutine returns a subsystem reply to a System Resource Controller (SRC) subsystem request. The format and content of the reply are determined by the subsystem and the requester, but must
start with a srchdr structure. This structure and all others required for subsystem communication with the SRC are defined in the /usr/include/spc.h file. The subsystem must reply with a pre-defined format and content for the following requests: START, STOP, STATUS, REFRESH, and TRACE. The START, STOP, REFRESH, and TRACE requests must be answered with a srcrep structure. The STATUS request must be answered with a reply in the form of a statbuf structure.

Note: The srcsrpy subroutine creates its own socket to send the subsystem reply packets.

Parameters

SRChdr Points to the reply address buffer as returned by the srcrqs subroutine.
PPacket Points to the reply packet. The first element of the reply packet is a srchdr structure. The cont element of the P Packet->srchdr structure is modified on returning from the srcsrpy subroutine. The second element of the reply packet should be a svreply structure, an array of statcode structures, or another format upon which the subsystem and the requester have agreed.
PPacketSize Specifies the number of bytes in the reply packet pointed to by the P Packet parameter. The PPacketSize parameter may be the size of a short, or it may be between the size of a srchdr structure and the SRCPKTMAX value, which is defined in the spc.h file.
Continued Indicates whether this reply is to be continued. If the Continued parameter is set to the constant END, no more reply packets are sent for this request. If the Continued parameter is set to CONTINUED, the second element of what is indicated by the P Packet parameter must be a svrreply structure, since the rtnmsg element of the svrreply structure is printed to standard output. For a status reply, the Continued parameter is set to STATCONTINUED, and the second element of what is pointed to by the P Packet parameter must be an array of statcode structures. If a STOP subsystem request is received, only one reply packet can be sent and the Continued parameter must be set to END. Other types of continuations, as determined by the subsystem and the requester, must be defined using positive values for the Continued parameter. Values other than the following must be used:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>END</td>
</tr>
<tr>
<td>1</td>
<td>CONTINUED</td>
</tr>
<tr>
<td>2</td>
<td>STATCONTINUED</td>
</tr>
</tbody>
</table>

Return Values
If the srcsrpy subroutine succeeds, it returns the value SRC_OK.

Error Codes
The srcsrpy subroutine fails if one or both of the following are true:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC_SOCK</td>
<td>There is a problem with SRC socket communications.</td>
</tr>
<tr>
<td>SRC_REPLYSZ</td>
<td>SRC reply size is invalid.</td>
</tr>
</tbody>
</table>

Examples
1. To send a STOP subsystem reply, enter:
   ```c
   struct srcrep return_packet;
   struct srchdr *srchdr;
   bzero(&return_packet,sizeof(return_packet));
   return_packet.svrreply.rtncode=SRC_OK;
   strcpy(return_packet.svrreply,"srctest");
   srcsrpy(srchdr,return_packet,sizeof(return_packet),END);
   ``
   This entry sends a message that the subsystem srctest is stopping successfully.
2. To send a START subserver reply, enter:
struct srcrep return_packet;
struct srchdr *srchdr;

bzero(&return_packet,sizeof(return_packet));
return_packet.svrreply.rtncode=SRC_SUBMSG;
strcpy(return_packet.svrreply.objname,"mysubserver");
strcpy(return_packet.svrreply.objtext,"The subserver, mysubserver, has been started");

srcsreply(srchdr,return_packet,sizeof(return_packet),END);
The resulting message indicates that the start subserver request was successful.

3. To send a status reply, enter:

int rc;
struct sockaddr addr;
int addrsz;
struct srcreq packet;
struct {
    struct srchdr srchdr;
    struct statcode statcode[10];
} status;
struct srchdr *srchdr;
struct srcreq packet;

/* grab the reply information from the SRC packet */
srchdr=srcrrqs(&packet);
bzero(&status.statcode[0].objname,

/* get SRC status header */
srcstathdr(status.statcode[0].objname,
    status.statcode[0].objtext);

/* send status packet(s) */
srcsreply(srchdr,&status,sizeof(status),STATCONTINUED);

srcsreply(srchdr,&status,sizeof(status),STATCONTINUED);

/* send final packet */
srcsreply(srchdr,&status,sizeof(struct srchdr),END);

This entry sends several status packets.

Files

/dev/.SRC-unix Specifies the location for temporary socket files.

Related Information

The srcrrqs ("srcrrqs Subroutine" on page 281) subroutine, srcsbuf ("srcsbuf Subroutine" on page 284) subroutine, srcsrqt ("srcsrqt Subroutine" on page 293) subroutine, srcstat ("srcstat Subroutine" on page 299) subroutine, srcstathdr ("srcstathdr Subroutine" on page 305) subroutine, srcstattxt ("srcstattxt Subroutine" on page 305) subroutine, srcstop ("srcstop Subroutine" on page 306) subroutine, srcsrtt ("srcsrtt Subroutine" on page 308) subroutine.

List of SRC Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
srcsrqt Subroutine

Purpose
Sends a request to a subsystem.

Library
System Resource Controller Library (libsrc.a)

Syntax
```c
#include <spc.h> srcsrqt(Host, SubsystemName, SubsystemPID,
RequestLength, SubsystemRequest, ReplyLength, ReplyBuffer, StartItAlso, Continued)
```
```
char * Host, * SubsystemName;
char * SubsystemRequest, * ReplyBuffer;
int SubsystemPID, StartItAlso, * Continued;
short RequestLength, * ReplyLength;
```

Description
The `srcsrqt` subroutine sends a request to a subsystem, waits for a response, and returns one or more replies to the caller. The format of the request and the reply is determined by the caller and the subsystem.

Note: The `srcsrqt` subroutine creates its own socket to send a request to the subsystem. The socket that this function opens remains open until an error or an end packet is received.

Two types of continuation are returned by the `srcsrqt` subroutine:

- **No continuation**: `ReplyBuffer->srchdr.continued` is set to the `END` constant.
- **Reply continuation**: `ReplyBuffer->srchdr.continued` is not set to the `END` constant, but to a positive value agreed upon by the calling process and the subsystem. The packet is returned to the caller.

Parameters

- **SubsystemPID**
  - The process ID of the subsystem.
- **Host**
  - Specifies the foreign host on which this subsystem request is to be sent. If the host is null, the request is sent to the subsystem on the local host. The local user must be running as “root”. The remote system must be configured to accept remote System Resource Controller requests. That is, the `srcmstr` daemon (see `/etc/inettab`) must be started with the `-r` flag and the `/etc/hosts.equiv` or `.rhosts` file must be configured to allow remote requests.
SubsystemName  Specifies the name of the subsystem to which this request is to be sent. You must specify a SubsystemName if you do not specify a SubsystemPID.

RequestLength  Specifies the length, in bytes, of the request to be sent to the subsystem. The maximum value in bytes for this parameter is 2000 bytes.

SubsystemRequest  Points to the subsystem request packet.

ReplyLength  Specifies the maximum length, in bytes, of the reply to be received from the subsystem. On return from the srcsrqt subroutine, the ReplyLength parameter is set to the actual length of the subsystem reply packet.

ReplyBuffer  Points to a buffer for the receipt of the reply packet from the subsystem.

StartItAlso  Specifies whether the subsystem should be started if it is nonactive. When nonzero, the System Resource Controller (SRC) attempts to start a nonactive subsystem, and then passes the request to the subsystem.

Continued  Specifies whether this call to the srcsrqt subroutine is a continuation of a previous request. If the Continued parameter is set to NEWREQUEST, a request for it is sent to the subsystem and the subsystem is notified that another response is expected. The calling process should never set Continued to any value other than NEWREQUEST. The last response from the subsystem will set Continued to END.

Return Values
If the srcsrqt subroutine is successful, the value SRC_OK is returned.

Error Codes
The srcsrqt subroutine fails if one or more of the following are true:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC_BADSOCK</td>
<td>The request could not be passed to the subsystem because of a socket failure.</td>
</tr>
<tr>
<td>SRC_CONT</td>
<td>The subsystem uses signals. The request cannot complete.</td>
</tr>
<tr>
<td>SRC_DMNA</td>
<td>The SRC daemon is not active.</td>
</tr>
<tr>
<td>SRC_INET_AUTHORIZED_HOST</td>
<td>The local host is not in the remote /etc/hosts.equiv file.</td>
</tr>
<tr>
<td>SRC_INET_INVALID_HOST</td>
<td>On the remote host, the local host is not known.</td>
</tr>
<tr>
<td>SRC_INVALID_USER</td>
<td>The user is not root or group system.</td>
</tr>
<tr>
<td>SRC_MMRY</td>
<td>An SRC component could not allocate the memory it needs.</td>
</tr>
<tr>
<td>SRC_NOCONTINUE</td>
<td>The Continued parameter was not set to NEWREQUEST, and no continuation is currently active.</td>
</tr>
<tr>
<td>SRC_NORPLY</td>
<td>The request timed out waiting for a response.</td>
</tr>
<tr>
<td>SRC_NSVR</td>
<td>The subsystem is not active.</td>
</tr>
<tr>
<td>SRC_REQLEN2BIG</td>
<td>The RequestLength is greater than the maximum 2000 bytes.</td>
</tr>
<tr>
<td>SRC_SOCK</td>
<td>There is a problem with SRC socket communications.</td>
</tr>
<tr>
<td>SRC_STPG</td>
<td>The request was not passed to the subsystem. The subsystem is stopping.</td>
</tr>
<tr>
<td>SRC_UDP</td>
<td>The SRC port is not defined in the /etc/services file.</td>
</tr>
<tr>
<td>SRC_UHOST</td>
<td>The foreign host is not known.</td>
</tr>
</tbody>
</table>

Examples
1. To request long subsystem status, enter:
   ```c
   int cont=NEWREQUEST;
   int rc;
   short replen;
   short reqlen;
   struct
   {   struct srchdr srchdr;
       struct statcode statcode[20];
   } statbuf;
   struct subreq subreq;
   ```
subreq.action=STATUS;
subreq.object=SUBSYSTEM;
subreq.parm1=LONGSTAT;
strcpy(subreq.objname,"srctest");
replen=sizeof(statbuf);
reqlen=sizeof(subreq);
rc=srcsrqt("MaryC", "srctest", 0, reqlen, &subreq, &replen, &statbuf, SRC_NO, &cont);

This entry gets long status of the subsystem srctest on the MaryC machine. The subsystem keeps sending status packets until statbuf.srchdr.cont=END.

2. To start a subserver, enter:
   int cont=NEWREQUEST;
   int rc;
   short replen;
   short reqlen;
   struct
     {
       struct srchdr srchdr;
       struct statcode statcode[20];
     } statbuf;
   struct subreq subreq;

   subreq.action=START;
   subreq.object=1234;
   replen=sizeof(statbuf);
   reqlen=sizeof(subreq);
   rc=srcsrqt("", ",", 987, reqlen, &subreq, &replen, &statbuf, SRC_NO, &cont);

   This entry starts the subserver with the code point of 1234, but only if the subsystem is already active.

3. To start a subserver and a subsystem, enter:
   int cont=NEWREQUEST;
   int rc;
   short replen;
   short reqlen;
   struct
     {
       struct srchdr srchdr;
       struct statcode statcode[20];
     } statbuf;
   struct subreq subreq;

   subreq.action=START;
   subreq.object=1234;
   replen=sizeof(statbuf);
   reqlen=sizeof(subreq);
   rc=srcsrqt("", ",", 987, reqlen, &subreq, &replen, &statbuf, SRC_YES, &cont);

   This entry starts the subserver with the code point of 1234. If the subsystem to which this subserver belongs is not active, the subsystem is started.

Files
/etc/services Defines sockets and protocols used for Internet services.
/dev/SRC Specifies the AF_UNIX socket file.
/dev./SRC-unix Specifies the location for temporary socket files.

Related Information
The srcrqs subroutine, srcbuf subroutine, srcsbuf subroutine, srcsrpy subroutine, srcstat subroutine.
srcsrqt_r Subroutine

Purpose
Sends a request to a subsystem.

Library
System Resource Controller Library (libsrc.a)

Syntax
#include <spc.h>
srsrcsrqt_r(Host, SubsystemName, SubsystemPID, RequestLength, 
    SubsystemRequest, ReplyLength, ReplyBuffer, StartItAlso, 
    Continued, SRCHandle)

Parameters

SubsystemPID
The process ID of the subsystem.

Host
Specifies the foreign host on which this subsystem request is to be sent. If the host is null, the request is sent to the subsystem on the local host.

Description
The srcsrqt_r subroutine sends a request to a subsystem, waits for a response and returns one or more replies to the caller. The format of the request and the reply is determined by the caller and the subsystem.

Note: For each NEWREQUEST, the srcsrqt_r subroutine creates its own socket to send a request to the subsystem. The socket that this function opens remains open until an error or an end packet is received.

This system is threadsafe and reentrant.

Two types of continuation are returned by the srcsrqt_r subroutine:

No continuation
ReplyBuffer->srchdr.continued is set to the END constant.

Reply continuation
ReplyBuffer->srchdr.continued is not set to the END constant, but to a positive value agreed upon by the calling process and the subsystem. The packet is returned to the caller.
SubsystemName

Specifies the name of the subsystem to which this request is to be sent. You must specify a SubsystemName if you do not specify a SubsystemPID.

RequestLength

Specifies the length, in bytes, of the request to be sent to the subsystem. The maximum length is 2000 bytes.

SubsystemRequest

Points to the subsystem request packet.

ReplyLength

Specifies the maximum length, in bytes, of the reply to be received from the subsystem. On return from the srcsrqt subroutine, the ReplyLength parameter is set to the actual length of the subsystem reply packet.

ReplyBuffer

Points to a buffer for the receipt of the reply packet from the subsystem.

StartItAlso

Specifies whether the subsystem should be started if it is nonactive. When nonzero, the System Resource Controller (SRC) attempts to start a nonactive subsystem, and then passes the request to the subsystem.

Continued

Specifies whether this call to the srcsrqt subroutine is a continuation of a previous request. If the Continued parameter is set to NEWREQUEST, a request for it is sent to the subsystem and the subsystem is notified that a response is expected. Under normal circumstances, the calling process should never set Continued to any value other than NEWREQUEST. The last response from the subsystem will set Continued to END. The caller must continue to call the srcsrqt_r subroutine until END is received. Otherwise, the socket will not be closed and the internal buffers freed. As an alternative, set Continued=SRC_CLOSE to discard the remaining data, close the socket, and free the internal buffers.

SRCHandle

Identifies a request and its associated responses. Set to NULL by the caller for a NEWREQUEST. The srcsrqt_r subroutine saves a value in SRCHandle to allow srcsrqt_r continuation calls to use the same socket and internal buffers. The SRCHandle parameter should not be changed by the caller except for NEWREQUESTs.

Return Values

If the srcsrqt_r subroutine is successful, the value SRC_OK is returned.

Error Codes

The srcsrqt_r subroutine fails and returns the corresponding error code if one of the following error conditions is detected:

SRC_BADSOCK

The request could not be passed to the subsystem because of a socket failure.

SRC_CONT

The subsystem uses signals. The request cannot complete.

SRC_DMNA

The SRC daemon is not active.

SRC_INET_AUTHORIZED_HOST

The local host is not in the remote /etc/hosts.equiv file.

SRC_INET_INVALID_HOST

On the remote host, the local host is not known.

SRC_INVALID_USER

The user is not root or group system.

SRC_MMRY

An SRC component could not allocate the memory it needs.

SRC_NOCONTINUE

The Continued parameter was not set to NEWREQUEST, and no continuation is currently active.

SRC_NORPLY

The request timed out waiting for a response.

SRC_NSVR

The subsystem is not active.

SRC_REQLEN2BIG

The RequestLength is greater than the maximum 2000 bytes.

SRC_SOCKET

There is a problem with SRC socket communications.

SRC_STPG

The request was not passed to the subsystem. The subsystem is stopping.

SRC_UDP

The SRC port is not defined in the /etc/services file.

SRC_UHOST

The foreign host is not known.
Examples

1. To request long subsystem status, enter:
   ```
   int cont=NEWREQUEST;
   int rc;
   short replen;
   short reqlen;
   char *handle;
   struct {
       struct srchdr srchdr;
       struct statcode statcode[20];
   } statbuf;
   struct subreq subreq;

   subreq.action=STATUS;
   subreq.object=SUBSYSTEM;
   subreq.parm1=LONGSTAT;
   strcpy(subreq.objname,"srctest");
   replen=sizeof(statbuf);
   reqlen=sizeof(subreq);
   rc=srcsrqt_r("MaryC", "srctest", 0, reqlen, &subreq, &replen, &statbuf, SRC_NO, &cont, &handle);
   
   This entry gets long status of the subsystem srctest on the MaryC machine. The subsystem keeps sending status packets until statbuf.srchdr.cont=END.
   ```

2. To start a subserver, enter:
   ```
   int cont=NEWREQUEST;
   int rc;
   short replen;
   short reqlen;
   struct {
       struct srchdr srchdr;
       struct statcode statcode[20];
   } statbuf;
   struct subreq subreq;

   subreq.action=START;
   subreq.object=1234;
   replen=sizeof(statbuf);
   reqlen=sizeof(subreq);
   rc=srcsrqt_r("", "", 987, reqlen, &subreq, &replen, &statbuf, SRC_NO, &cont, &handle);
   
   This entry starts the subserver with the code point of 1234, but only if the subsystem is already active.
   ```

3. To start a subserver and a subsystem, enter:
   ```
   int cont=NEWREQUEST;
   int rc;
   short replen;
   short reqlen;
   char *handle;
   struct {
       struct srchdr srchdr;
       struct statcode statcode[20];
   } statbuf;
   struct subreq subreq;

   subreq.action=START;
   subreq.object=1234;
   replen=sizeof(statbuf);
   reqlen=sizeof(subreq);
   rc=srcsrqt("", "", 987, reqlen, &subreq, &replen, &statbuf, SRC_YES, &cont, &handle);
   ```
This entry starts the subserver with the code point of 1234. If the subsystem to which this subserver belongs is not active, the subsystem is started.

Files

/etc/services Defines sockets and protocols used for Internet services.
/dev/SRC Specifies the AF_UNIX socket file.
/dev/.SRC-unix Specifies the location for temporary socket files.

Related Information

The src_err_msg_r subroutine sends a short status request to the System Resource Controller (SRC) and returns status for one or more subsystems to the caller.

The src_sbuf_r subroutine sends a short status request to the System Resource Controller (SRC) and returns status for one or more subsystems to the caller.

The srcrrqs_r subroutine sends a short status request to the System Resource Controller (SRC) and returns status for one or more subsystems to the caller.

The srcstat_r subroutine sends a short status request to the System Resource Controller (SRC) and returns status for one or more subsystems to the caller.

The srcstattxt_r subroutine sends a short status request to the System Resource Controller (SRC) and returns status for one or more subsystems to the caller.

List of SRC Subroutines

In AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Programming Subsystem Communication with the SRC

In AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

System Resource Controller (SRC) Overview for Programmers

In AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

srcstat Subroutine

Purpose

Gets short status on one or more subsystems.

Library

System Resource Controller Library (libsrc.a)

Syntax

```
#include <spc.h>

int srcstat(Host,
    SubsystemName,SubsystemPID, ReplyLength, StatusReply,Continued)

char * Host, *SubsystemName;
int SubsystemPID, *Continued;
short * ReplyLength;
void * StatusReply;
```

Description

The srcstat subroutine sends a short status request to the System Resource Controller (SRC) and returns status for one or more subsystems to the caller.
Parameters

**Host**
Specifies the foreign host on which this status action is requested. If the host is null, the
status request is sent to the SRC on the local host. The local user must be running as
"root". The remote system must be configured to accept remote System Resource
Controller requests. That is, the srcmstr daemon (see /etc/inittab) must be started with the
-r flag and the /etc/hosts.equiv or .rhosts file must be configured to allow remote
requests.

**SubsystemName**
Specifies the name of the subsystem on which to get short status. To get status of all
subsystems, use the SRCALLSUBSYS constant. To get status of a group of subsystems,
the SubsystemName parameter must start with the SRCGROUP constant, followed by the
name of the group for which you want status appended. If you specify a null
SubsystemName parameter, you must specify a SubsystemPID parameter.

**SubsystemPID**
Specifies the PID of the subsystem on which to get status as returned by the srcstat
subroutine. You must specify the SubsystemPID parameter if multiple instances of the
subsystem are active and you request a long subsystem status or subserver status. If you
specify a null SubsystemPID parameter, you must specify a SubsystemName parameter.

**ReplyLength**
Specifies size of a srchdr structure plus the number of statcode structures times the size
of one statcode structure. On return from the srcstat subroutine, this value is updated.

**StatusReply**
Specifies a pointer to a structure containing first element as struct srchdr and secondary
element as struct statcode (both defined in spc.h file) array that receives the status reply
for the requested subsystem. The first element of the returned statcode array contains the
status title line. The number of statcode structures array items depends on the number of
subsystems user queried.

**Continued**
Specifies whether this call to the srcstat subroutine is a continuation of a previous status
request. If the Continued parameter is set to NEWSREQUEST, a request for short
subsystem status is sent to the SRC and srcstat waits for the first status response. The
calling process should never set Continued to a value other than NEWSREQUEST. The last
response for the SRC sets Continued to END.

Return Values
If the srcstat subroutine succeeds, it returns a value of 0. An error code is returned if the subroutine is
unsuccessful.

Error Codes
The srcstat subroutine fails if one or more of the following are true:

- **SRC_DMNA**
The SRC daemon is not active.
- **SRC_INETAUTHORIZED_HOST**
The local host is not in the remote /etc/hosts.equiv file.
- **SRC_INETINVALID_HOST**
On the remote host, the local host is not known.
- **SRC_INVALID_USER**
The user is not root or group system.
- **SRC_MMRY**
An SRC component could not allocate the memory it needs.
- **SRC_NOCONTINUE**
Continued was not set to NEWSREQUEST and no continuation is
currently active.
- **SRC_NORPLY**
The request timed out waiting for a response.
- **SRC_SOCK**
There is a problem with SRC socket communications.
- **SRC_UDP**
The SRC port is not defined in the /etc/services file.
- **SRC_UHOST**
The foreign host is not known.

Examples
1. To request the status of a subsystem, enter:
   ```c
   intcont=NEWREQUEST;
   struct {
      struct srchdr srchdr
      struct statcode statcode[6];
   ```
2. To request the status of all subsystems, enter:

```c
int cont=NEWREQUEST;
struct {
    struct srchdr srchdr;
    struct statcode statcode[80];
} status;
short rep_len=sizeof(status);

srcstat("",SRCALLSUBSYS,0,&rep_len,&status,&cont);
```
This entry requests short status of all subsystems on the local machine.

3. To request the status for a group of subsystems, enter:

```c
int cont=NEWREQUEST;
struct struct {
    struct srchdr srchdr;
    struct statcode statcode[30];
} status;
short rep_len=sizeof(status), rep_num;
char subsysname[30];

strcpy(subsysname,SRCGROUP);
strcat(subsysname,"tcpip");
srctest("",subsysname,0,&rep_len,&status, &cont);
```
rep_num = (rep_len - sizeof(struct srchdr)) / sizeof(struct statcode);

```
for (i = 0; i < rep_num; i++)
    printf("%d %d %s %s\n",
        status.statcode[i].objtype, status.statcode[i].status,
        status.statcode[i].objname, status.statcode[i].objtext);
```
This entry requests short status of all members of the subsystem group tcpip on the local machine, and displays the query results on stdout.

Files

/`etc/services` Defines the sockets and protocols used for Internet services.
/`dev/SRC` Specifies the `AF_UNIX` socket file.
/`dev/.SRC-unix` Specifies the location for temporary socket files.

Related Information

The `srcrrqs` subroutine, `srcbuf` subroutine, `srsrpy` subroutine, `srcsrqt` subroutine, `srcsrthdr` subroutine, `srcstathdr` subroutine, `srcstattxt` subroutine, `srcstop` subroutine, `srcstrt` subroutine.

List of SRC Subroutines | Programming Subsystem Communication with the SRC | System Resource Controller (SRC) Overview for Programmers | Writing and Debugging Programs

Chapter 1. Base Operating System (BOS) Runtime Services (Q-Z) 301
srcstat_r Subroutine

Purpose
Gets short status on a subsystem.

Library
System Resource Controller Library (libsrc.a)

Syntax
#include <spc.h>

int srcstat_r(Host, SubsystemName, SubsystemPID, ReplyLength,
              StatusReply, Continued, SRCHandle)
char *Host, *SubsystemName;
pid_t SubsystemPID;
int *Continued;
short *ReplyLength;
struct statrep *StatusReply;
char **SRCHandle;

Description
The srcstat_r subroutine sends a short status request to the System Resource Controller (SRC) and returns status for one or more subsystems to the caller. This subroutine is threadsafe and reentrant.

Parameters

Host Specifies the foreign host on which this status action is requested. If the host is null, the status request is sent to the SRC on the local host.

SubsystemName Specifies the name of the subsystem on which to get short status. To get status of all subsystems, use the SRCALLSUBSYS constant. To get status of a group of subsystems, the SubsystemName parameter must start with the SRCGROUP constant, followed by the name of the group for which you want status appended. If you specify a null SubsystemName parameter, you must specify a SubsystemPID parameter.

SubsystemPID Specifies the PID of the subsystem on which to get status as returned by the srcstat_r subroutine. You must specify the SubsystemPID parameter if multiple instances of the subsystem are active and you request a long subsystem status or subserver status. If you specify a null SubsystemPID parameter, you must specify a SubsystemName parameter.

ReplyLength Specifies size of a srchdr structure plus the number of statcode structures times the size of one statcode structure. On return from the srcstat_r subroutine, this value is updated.

StatusReply Specifies a pointer to a statrep code structure containing a statcode array that receives the status reply for the requested subsystem. The first element of the returned statcode array contains the status title line. The statcode structure is defined in the spc.h file.

Continued Specifies whether this call to the srcstat_r subroutine is a continuation of a previous status request. If the Continued parameter is set to NEWREQUEST, a request for short subsystem status is sent to the SRC and srcstat_r waits for the first status response. During NEWREQUEST processing, the srcstat_r subroutine opens a socket, mallocs internal buffers, and saves a value in SRCHandle. In normal circumstances, the calling process should never set Continued to a value other than NEWREQUEST. When the srcstat_r subroutine returns with Continued=STATCONTINUED, call srcstat_r without changing the Continued and SRCHandle parameters to receive additional data. The last response from the SRC sets Continued to END. The caller must continue to call srcstat_r until END is received. Otherwise, the socket will not be closed and the internal buffers freed. As an alternative, call srcstat_r with Continued=STATCONTINUED to discard the remaining data, close the socket, and free the internal buffers.
Identifies a request and its associated responses. Set to NULL by the caller for a **NEWREQUEST**. The `srcstat_r` subroutine saves a value in `SRCHandle` to allow subsequent `srcstat_r` calls to use the same socket and internal buffers. The `SRCHandle` parameter should not be changed by the caller except for **NEWREQUESTs**.

**Return Values**
If the `srcstat_r` subroutine succeeds, it returns a value of 0. An error code is returned if the subroutine is unsuccessful.

**Error Codes**
The `srcstat_r` subroutine fails and returns the corresponding error code if one of the following error conditions is detected:

- **SRC_DMNA**: The SRC daemon is not active.
- **SRC_INET_AUTHORIZED_HOST**: The local host is not in the remote `/etc/hosts.equiv` file.
- **SRC_INET_INVALID_HOST**: On the remote host, the local host is not known.
- **SRC_INVALID_USER**: The user is not root or group system.
- **SRC_MMRY**: An SRC component could not allocate the memory it needs.
- **SRC_NOCONTINUE**: `Continued` was not set to **NEWREQUEST** and no continuation is currently active.
- **SRC_NORPLY**: The request timed out waiting for a response.
- **SRC.SOCK**: There is a problem with SRC socket communications.
- **SRC_UHOST**: The foreign host is not known.

**Examples**
1. To request the status of a subsystem, enter:
   ```c
   int cont=NEWREQUEST;
   struct statcode statcode[20];
   short replen=sizeof(statcode);
   char *handle;

   srcstat_r("MaryC","srctest",0,&replen,statcode,&cont,&handle);
   
   This entry requests short status of all instances of the subsystem srctest on the MaryC machine.
   ```
2. To request the status of all subsystems, enter:
   ```c
   int cont=NEWREQUEST;
   struct statcode statcode[20];
   short replen=sizeof(statcode);
   char *handle;

   srcstat_r("","SRCALLSUBSYS",0,&replen,statcode,&cont,&handle);
   
   This entry requests short status of all subsystems on the local machine.
   ```
3. To request the status for a group of subsystems, enter:
   ```c
   int cont=NEWREQUEST;
   struct statcode statcode[20];
   short replen=sizeof(statcode);
   char subsysname[30];
   char *handle;

   strcpy(subsysname,SRCGROUP);
   strcat(subsysname,"tcpip");
   srcstat_r("",subsysname,0,&replen,statcode,&cont,&handle);
   
   This entry requests short status of all members of the subsystem group tcpip on the local machine.
Files

/etc/services
Defines the sockets and protocols used for Internet services.
/dev/SRC
Specifies the `AF_UNIX` socket file.
/dev/.SRC-unix
Specifies the location for temporary socket files.

Related Information
The src_errno_r subroutine, srcbuf_r subroutine, srcsrq_r subroutine, srcsrpy_r subroutine, srcstat_r subroutine, srcstop_r subroutine, and srcstattxt_r subroutine.

srctstathdr Subroutine

Purpose
Gets the title line of the System Resource Controller (SRC) status text.

Library
System Resource Controller Library (`libsrc.a`)

Syntax
```c
void srctstathdr (char *Title1, char *Title2);
```

Description
The `srctstathdr` subroutine retrieves the title line, or header, of the SRC status text.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Title1</code></td>
<td>Specifies the <code>objname</code> field of a <code>statcode</code> structure. The subsystem name title is placed here.</td>
</tr>
<tr>
<td><code>Title2</code></td>
<td>Specifies the <code>objtext</code> field of a <code>statcode</code> structure. The remaining titles are placed here.</td>
</tr>
</tbody>
</table>

Return Values
The subsystem name title is returned in the `Title1` parameter. The remaining titles are returned in the `Title2` parameter.

Related Information
The srcrrqs subroutine, srcbuf subroutine, srcsrq subroutine, srcsrpy subroutine, srcstat subroutine, srcstattxt subroutine, srcstop subroutine, and srcstattxt subroutine.

List of SRC Subroutines, Programming Subsystem Communication with the SRC, System Resource Controller (SRC) Overview for Programmers in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
srcstattxt Subroutine

Purpose
Gets the System Resource Controller (SRC) status text representation for a status code.

Library
System Resource Controller Library (libsrc.a)

Syntax
char *srcstattxt ( StatusCode )
short StatusCode;

Description
The srcstattxt subroutine, given an SRC status code, gets the text representation and returns a pointer to this text.

Parameters
StatusCode Specifies an SRC status code to be translated into meaningful text.

Return Values
The srcstattxt subroutine returns a pointer to the text representation of a status code.

Related Information
The srcrqs ("srcrqs Subroutine" on page 281) subroutine, srcbuf ("srcbuf Subroutine" on page 284) subroutine, srcspsy ("srcspsy Subroutine" on page 290) subroutine, srcsrqt ("srcsrqt Subroutine" on page 293) subroutine, srcstat ("srcstat Subroutine" on page 299) subroutine, srcstattdr ("srcstattdr Subroutine" on page 304) subroutine, srccstop ("srccstop Subroutine" on page 306) subroutine, srcstrt ("srcstrt Subroutine" on page 308) subroutine.

srcstattxt_r Subroutine

Purpose
Gets the status text representation for an SRC status code.

Library
System Resource Controller Library (libsrc.a)

Syntax
#include <spc.h>

char *srcstattxt_r ( StatusCode, Text )
short StatusCode;
char *Text;
Description
The srcstattxt_r subroutine, given an SRC status code, gets the text representation and returns it in a caller-supplied buffer. This routine is threadsafe and reentrant.

Parameters
StatusCode  Specifies an SRC status code to be translated into meaningful text.
Text  Points to a caller-supplied buffer where the text will be returned. If the length of the text is unknown, the maximum text length can be used when allocating the buffer. The maximum text length is SRC_STAT_MAX in /usr/include/spc.h (64 bytes).

Return Values
Upon successful completion, the srcstattxt_r subroutine returns the address of the caller-supplied buffer. Otherwise, no text is returned and the subroutine returns NULL.

Related Information
The src_err_msg_r ("src_err_msg_r Subroutine" on page 281), srcsbuf_r ("srcsbuf_r Subroutine" on page 287), srcsrqt_r ("srcsrqt_r Subroutine" on page 296), srcrrqs_r ("srcrrqs_r Subroutine" on page 283), and srcstat_r ("srcstat_r Subroutine" on page 302) subroutines.

List of SRC Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Programming Subsystem Communication with the SRC in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

System Resource Controller (SRC) Overview for Programmers in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

srcstop Subroutine

Purpose
Stops a System Resource Controller (SRC) subsystem.

Library
System Resource Controller Library (libsrc.a)

Syntax
#include <spc.h>

srcstop(Host, SubsystemName, SubsystemPID, StopType)
srcstop(ReplyLength, ServerReply, StopFrom)

char *Host, *SubsystemName;
int SubsystemPID, StopFrom;
short StopType, *ReplyLength;
struct srcrep *ServerReply;

Description
The srcstop subroutine sends a stop subsystem request to a subsystem and waits for a stop reply from the System Resource Controller (SRC) or the subsystem. The srcstop subroutine can only stop a subsystem that was started by the SRC.
Parameters

*Host*
Specifies the foreign host on which this stop action is requested. If the host is the null value, the request is sent to the SRC on the local host. The local user must be running as "root". The remote system must be configured to accept remote System Resource Controller requests. That is, the srcmstr daemon (see `/etc/inittab`) must be started with the `-r` flag and the `/etc/hosts.equiv` or `.rhosts` file must be configured to allow remote requests.

*SubsystemName*
Specifies the name of the subsystem to stop.

*SubsystemPID*
Specifies the process ID of the system to stop as returned by the srcstrt subroutine. If you specify a null `SubsystemPID` parameter, you must specify a `SubsystemName` parameter.

*StopType*
Specifies the type of stop requested of the subsystem. If this parameter is null, a normal stop is assumed. The `StopType` parameter must be one of the following values:

**CANCEL**
Requires a quick stop of the subsystem. The subsystem is sent a `SIGTERM` signal. After the wait time defined in the subsystem object, the SRC issues a `SIGKILL` signal to the subsystem. This waiting period allows the subsystem to clean up all its resources and terminate. The stop reply is returned by the SRC.

**FORCE**
Requests a quick stop of the subsystem and all its subservers. The stop reply is returned by the SRC for subsystems that use signals and by the subsystem for other communication types.

**NORMAL**
Requests the subsystem to terminate after all current subsystem activity has completed. The stop reply is returned by the SRC for subsystems that use signals and by the subsystem for other communication types.

*ReplyLength*
Specifies the maximum length, in bytes, of the stop reply. On return from the `srcstop` subroutine, this field is set to the actual length of the subsystem reply packet received.

*ServerReply*
Points to an `svrreply` structure that will receive the subsystem stop reply.

*StopFrom*
Specifies whether the `srcstop` subroutine is to display stop results to standard output. If the `StopFrom` parameter is set to `SSHELL`, the stop results are displayed to standard output and the `srcstop` subroutine returns successfully. If the `StopFrom` parameter is set to `SDAEMON`, the stop results are not displayed to standard output, but are passed back to the caller.

Return Values

Upon successful completion, the `srcstop` subroutine returns `SRC_OK` or `SRC_STPOK`.

Error Codes

The `srcstop` subroutine fails if one or more of the following are true:

- **SRC_BADFSIG**
The stop force signal is an invalid signal.
- **SRC_BADNSIG**
The stop normal signal is an invalid signal.
- **SRC_BADSOCK**
The stop request could not be passed to the subsystem on its communication socket.
- **SRC_DMNA**
The SRC daemon is not active.
- **SRC_INET_AUTHORIZED_HOST**
The local host is not in the remote `/etc/hosts.equiv` file.
- **SRC_INET_INVALID_HOST**
On the remote host, the local host is not known.
- **SRC_INVALID_USER**
The user is not root or group system.
- **SRC_MMMRY**
An SRC component could not allocate the memory it needs.
- **SRC_NORPLY**
The request timed out waiting for a response.
- **SRC_NOTROOT**
The SRC daemon is not running as root.
- **SRC.SOCK**
There is a problem with SRC socket communications.
- **SRC_STPG**
The request was not passed to the subsystem. The subsystem is stopping.
The subsystem is unknown to the SRC daemon.
The remote SRC port is not defined in the /etc/services file.
The foreign host is not known.
Invalid parameter passed.

Examples
1. To stop all instances of a subsystem, enter:
   ```c
   int rc;
   struct svrreply svrreply;
   short replen=sizeof(svrreply);
   rc=srcstop("MaryC","srctest",0,FORCE,&replen,&svrreply,SDAEMON);
   
   This request stops a subsystem with a stop type of FORCE for all instances of the subsystem srctest on the MaryC machine and does not print a message to standard output about the status of the stop.
   
   2. To stop a single instance of a subsystem, enter:
      ```c
      struct svrreply svrreply;
      short replen=sizeof(svrreply);
      rc=srcstop("","",999,CANCEL,&replen,&svrreply,SSHELL);
      
      This request stops a subsystem with a stop type of CANCEL, with the process ID of 999 on the local machine and prints a message to standard output about the status of the stop.
      
Files
/etc/services Defines sockets and protocols used for Internet services.
/dev/SRC Specifies the AF_UNIX socket file.
/dev/.SRC-unix Specifies the location for temporary socket files.

Related Information
The srcrrqs ("srcrrqs Subroutine" on page 281) subroutine, srcsbuf ("srcsbuf Subroutine" on page 284) subroutine, srcspsy ("srcspsy Subroutine" on page 290) subroutine, srcsreqt ("srcsreqt Subroutine" on page 293) subroutine, srcstat ("srcstat Subroutine" on page 299) subroutine, srcstathtdr ("srcstathtdr Subroutine" on page 304) subroutine, srcstattxt ("srcstattxt Subroutine" on page 305) subroutine, srcstrt ("srcstrt Subroutine") subroutine.

List of SRC Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Programming Subsystem Communication with the SRC in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

System Resource Controller (SRC) Overview for Programmers in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

srcstrt Subroutine

Purpose
Starts a System Resource Controller (SRC) subsystem.

Library
System Resource Controller Library (libsrc.a)
#include<spc.h>

srcstrt (Host, SubsystemName, Environment, Arguments, Restart, StartFrom)

char * Host, * SubsystemName;
char * Environment, * Arguments;
unsigned int Restart;
int StartFrom;

Description
The srcstrt subroutine sends a start subsystem request packet and waits for a reply from the System Resource Controller (SRC).

Parameters

Host Specifies the foreign host on which this start subsystem action is requested. If the host is null, the request is sent to the SRC on the local host. The local user must be running as "root". The remote system must be configured to accept remote System Resource Controller requests. That is, the srcmstr daemon (see /etc/inittab) must be started with the -r flag and the /etc/hosts.equiv or .rhosts file must be configured to allow remote requests.

SubsystemName Specifies the name of the subsystem to start.

Environment Specifies a string that is placed in the subsystem environment when the subsystem is executed. The combined values of the Environment and Arguments parameters cannot exceed a maximum of 2400 characters. Otherwise, the srcstrt subroutine will fail. The environment string is parsed by the SRC according to the same rules used by the shell. For example, quoted strings are passed as a single Environment value, and blanks outside a quoted string delimit each environment value.

Arguments Specifies a string that is passed to the subsystem when the subsystem is executed. The string is parsed from the command line and appended to the command line arguments from the subsystem object class. The combined values of the Environment and Arguments parameters cannot exceed a maximum of 2400 characters. Otherwise, the srcstrt subroutine will fail. The command argument is parsed by the SRC according to the same rules used by the shell. For example, quoted strings are passed as a single argument, and blanks outside a quoted string delimit each argument.

Restart Specifies override on subsystem restart. If the Restart parameter is set to SRCNO, the subsystem's restart definition from the subsystem object class is used. If the Restart parameter is set to SRCYES, the restart of a subsystem is not attempted if it terminates abnormally.

StartFrom Specifies whether the srcstrt subroutine is to display start results to standard output. If the StartFrom parameter is set to SSHELL, the start results are displayed to standard output, and the srcstrt subroutine always returns successfully. If the StartFrom parameter is set to SDAEMON, the start results are not displayed to standard output but are passed back to the caller.

Return Values
When the StartFrom parameter is set to SSHELL, the srcstrt subroutine returns the value SRC_OK. Otherwise, it returns the subsystem process ID.

Error Codes
The srcstrt subroutine fails if any of the following are true:

SRC_AUDITID The audit user ID is invalid.
The SRC daemon is not active.
The subsystem could not be forked and executed.
The local host is not in the remote /etc/hosts.equiv file.
On the remote host, the local host is not known.
The user is not root or group system.
The subsystem standard input file could not be established.
An SRC component could not allocate the memory it needs.
The subsystem message queue could not be created.
Multiple instance of the subsystem are not allowed.
The request timed out waiting for a response.
The subsystem standard output file could not be established.
A pipe could not be established for the subsystem.
The subsystem standard error file could not be established.
The subsystem communication socket could not be created.
The system user ID is invalid.
There is a problem with SRC socket communications.
The subsystem is unknown to the SRC daemon.
The SRC port is not defined in the /etc/services header file.
The foreign host is not known.

Examples
1. To start a subsystem passing the Environment and Arguments parameters, enter:
   
   rc=srcstrt("","srctest","HOME=/tmpTERM=ibm6155",
   "-z\"thezflagargument\"",SRC_YES,SSHELL);
   
   This starts the srctest subsystem on the local host, placing HOME=/tmp, TERM=ibm6155 in the environment and using -z and thezflagargument as two arguments to the subsystem. This also displays the results of the start command to standard output and allows the SRC to restart the subsystem should it end abnormally.

2. To start a subsystem on a foreign host, enter:
   
   rc=srcstrt("MaryC","srctest","",",",SRC_NO,SDAEMON);
   
   This starts the srctest subsystem on the MaryC machine. This does not display the results of the start command to standard output and does not allow the SRC to restart the subsystem should it end abnormally.

Files
/etc/services Defines sockets and protocols used for Internet services.
/dev/SRC Specifies the AF_UNIX socket file.
/dev/.SRC-unix Specifies the location for temporary socket files.

Related Information
The srcrrqs ("srcrrqs Subroutine" on page 281) subroutine, srcbuf ("srcbuf Subroutine" on page 284) subroutine, srcrpy ("srcrpy Subroutine" on page 290) subroutine, srcsqt ("srcsqt Subroutine" on page 293) subroutine, srcstat ("srcstat Subroutine" on page 299) subroutine, srcstatshr ("srcstatshr Subroutine" on page 304) subroutine, srcstatxt ("srcstatxt Subroutine" on page 305) subroutine, srcstop ("srcstop Subroutine" on page 306) subroutine.

List of SRC Subroutines Programming Subsystem Communication with the SRC System Resource Controller (SRC) Overview for Programmers in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
ssignal or gsignal Subroutine

Purpose
Implements a software signal facility.

Library
Standard C Library (libc.a)

Syntax
#include <signal.h>

void (*ssignal (Signal, Action))()
int Signal;
void (*Action)();
int gsignal (Signal)
int Signal;

Description
Attention: Do not use the ssignal or gsignal subroutine in a multithreaded environment.

The ssignal and gsignal subroutines implement a software facility similar to that of the signal and kill subroutines. However, there is no connection between the two facilities. User programs can use the ssignal and gsignal subroutines to handle exceptional processing within an application. The signal subroutine and related subroutines handle system-defined exceptions.

The software signals available are associated with integers in the range 1 through 16. Other values are reserved for use by the C library and should not be used.

The ssignal subroutine associates the procedure specified by the Action parameter with the software signal specified by the Signal parameter. The gsignal subroutine raises the Signal, causing the procedure specified by the Action parameter to be taken.

The Action parameter is either a pointer to a user-defined subroutine, or one of the constants SIG_DFL (default action) and SIG_IGN (ignore signal). The ssignal subroutine returns the procedure that was previously established for that signal. If no procedure was established before, or if the signal number is illegal, then the ssignal subroutine returns the value of SIG_DFL.

The gsignal subroutine raises the signal specified by the Signal parameter by doing the following:
• If the procedure for the Signal parameter is SIG_DFL, the gsignal subroutine returns a value of 0 and takes no other action.
• If the procedure for the Signal parameter is SIG_IGN, the gsignal subroutine returns a value of 1 and takes no other action.
• If the procedure for the Signal parameter is a subroutine, the Action value is reset to the SIG_DFL procedure and the subroutine is called, with the Signal value passed as its parameter. The gsignal subroutine returns the value returned by the signal-handling routine.
• If the Signal parameter specifies an illegal value or if no procedure is specified for that signal, the gsignal subroutine returns a value of 0 and takes no other action.

Parameters
Signal Specifies a signal.
Action Specifies a procedure.
Related Information

The kill or killpg subroutine, signal subroutine. (sigaction, sigvec, or signal Subroutine” on page 203)” subroutine.

statacl or fstatacl Subroutine

Purpose

Retrieves the AIXC ACL type access control information for a file.

Library

Standard C Library (libc.a)

Syntax

```c
#include <sys/acl.h>
#include <sys/stat.h>

int statacl (char *Path, int Command, struct acl *ACL, int ACLSize);

int fstatacl (int FileDescriptor, int Command, struct acl *ACL, int ACLSize);
```

Description

The statacl and fstatacl subroutines return the access control information for a file system object if the ACL associated is of AIXC type. If the ACL associated is of different type or if the underlying physical file system does not support AIXC ACL type, error could be returned by these interfaces. It is recommended strongly that applications stop using these interfaces and instead make use of aclx_get or aclx_fget subroutines to get the ACL.

Parameters

- **Path**
  - Specifies a pointer to the path name of a file.
- **FileDescriptor**
  - Specifies the file descriptor of an open file.
- **Command**
  - Specifies the mode of the path interpretation for Path, specifically whether to retrieve information about a symbolic link or mount point. Valid values for the Command parameter are defined in the stat.h file and include:
    - *STX_LINK*
    - *STX_MOUNT*
    - *STX_NORMAL*
ACL

Specifies a pointer to a buffer to contain the AIXC-type Access Control List (ACL) of the file system object. The format of an AIXC ACL is defined in the `sys/acl.h` file and includes the following members:

**acl_len**

Size of the Access Control List (ACL).

*Note:* The entire ACL for a file cannot exceed one memory page (4096 bytes).

**acl_mode**

File mode.

*Note:* The valid values for the `acl_mode` are defined in the `sys/mode.h` file.

**u_access**

Access permissions for the file owner.

**g_access**

Access permissions for the file group.

**o_access**

Access permissions for default class `others`.

**acl_ext[ ]**

An array of the extended entries for this access control list.

The members for the base ACL (owner, group, and others) may contain the following bits, which are defined in the `sys/access.h` file:

**R_ACC**

Allows read permission.

**W_ACC**

Allows write permission.

**X_ACC**

Allows execute or search permission.

**ACLSize**

Specifies the size of the buffer to contain the ACL. If this value is too small, the first word of the ACL is set to the size of the buffer needed.

**Return Values**

On successful completion, the `statacl` and `fstatacl` subroutines return a value of 0. Otherwise, a value of -1 is returned and the `errno` global variable is set to indicate the error.

**Error Codes**

The `statacl` subroutine fails if one or more of the following are true:

**ENOTDIR**

A component of the `Path` prefix is not a directory.

**ENOENT**

A component of the `Path` does not exist or has the `disallow truncation` attribute (see the `ulimit` subroutine).

**ENOENT**

The `Path` parameter was null.

**EACCESE**

Search permission is denied on a component of the `Path` prefix.

**EFAULT**

The `Path` parameter points to a location outside of the allocated address space of the process.

**ESTALE**

The process’ root or current directory is located in a virtual file system that has been unmounted.

**ELOOP**

Too many symbolic links were encountered in translating the `Path` parameter.

**ENOENT**

A symbolic link was named, but the file to which it refers does not exist.

**ENAMETOOLONG**

A component of the `Path` parameter exceeded 255 characters, or the entire `Path` parameter exceeded 1023 characters.
The `fstatacl` subroutine fails if the following is true:

EBADF    The file descriptor `FileDescriptor` is not valid.

The `statacl` or `fstatacl` subroutine fails if one or more of the following are true:

EFAULT    The `ACL` parameter points to a location outside of the allocated address space of the process.
EINVAL    The `Command` parameter is not a value of `STX_LINK`, `STX_MOUNT`, `STX_NORMAL`.
ENOSPC    The `ACLSize` parameter indicates the buffer at `ACL` is too small to hold the Access Control List. In this case, the first word of the buffer is set to the size of the buffer required.
EIO        An I/O error occurred during the operation.

If Network File System (NFS) is installed on your system, the `statacl` and `fstatacl` subroutines can also fail if the following is true:

ETIMEDOUT The connection timed out.

Related Information
The `chac[1]` subroutine, `stat` (*`statx`, `stat`, `lstat`, `fstatx`, `fstat`, `fullstat`, `ffullstat`, `stat64`, `lstat64`, `fstat64`, `stat64x`, `fstat64x`, or `lstat64x` Subroutine* on page 318) subroutine.

The `acl.chg` subroutine, `acl.get` subroutine, `acl.put` subroutine, `acl.set` subroutine.

The `aclx_get Subroutine`, `aclx_put Subroutine`.

The `aclget` command, `aclput` command, `chmod` command.

List of Security and Auditing Subroutines and Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

statea Subroutine

Purpose
Provides information about an extended attribute.

Syntax

```c
#include <sys/ea.h>

int statea(const char *path, const char *name, struct stat64x *buffer)
int fstatea(int filedes, const char *name, struct stat64x *buffer)
int lstatea(const char *path, const char *name, struct stat64x *buffer)
```

Description
Extended attributes are name:value pairs associated with the file system objects (such as files, directories, and symlinks). They are extensions to the normal attributes that are associated with all objects in the file system (that is, the `stat(2)` data).

The `statea` subroutine gets information about the extended attribute name `name` associated with the file system object specified by `path`. The `fstatea` subroutine is identical to `statea`, except that it takes a file descriptor instead of a path. The `lstatea` subroutine is identical to `statea`, except, in the case of a symbolic link, the link itself is interrogated rather than the file that it refers to.
The `statea` subroutine uses a `stat64x` structure to return the information. Note that all values in this structure are 64-bit, including the devices and size. A normal `struct stat` cannot be passed to `statea`. For more information, see the "statx, stat, lstat, fstatx, fstat, fullstat, ffullstat, stat64, lstat64, fstat64, stat64x, fstat64x, or lstat64x Subroutine" on page 318.

**Parameters**

- **path** The path name of the file.
- **name** The name of the extended attribute. An extended attribute name is a NULL-terminated string.
- **buffer** A pointer to the stat structure in which information is returned.
- **filedes** A file descriptor for the file.

**Return Values**

If the `statea` subroutine succeeds, 0 is returned. Upon failure, -1 is returned and `errno` is set appropriately.

**Error Codes**

- **EACCES** Caller lacks read permission on the base file, or lacks the appropriate ACL privileges for named attribute lookup.
- **EFAULT** A bad address was passed for `path`, `name`, or `buffer`.
- **EFORMAT** File system is capable of supporting EAs, but EAs are disabled.
- **EINVAL** A path-like name should not be used (such as `zml/file`, `.`, and `..`).
- **ENAMETOOLONG** The `path` or `name` value is too long.
- **ENOATTR** No attribute named `name` is present.
- **ENOTSUP** Extended attributes are not supported by the file system.

**Related Information**

- `getea Subroutine`, `listea Subroutine` ["removeea Subroutine" on page 53, "setea Subroutine" on page 166, "statx, stat, lstat, fstatx, fstat, fullstat, ffullstat, stat64, lstat64, fstat64, stat64x, fstat64x, or lstat64x Subroutine" on page 318].

**statfs, fstatfs, statfs64, fstatfs64, or ustat Subroutine**

**Purpose**

Gets file system statistics.

**Library**

Standard C Library (`libc.a`)

**Syntax**

```c
#include <sys/statfs.h>

int statfs (Path, StatusBuffer)
char *Path;
struct statfs *StatusBuffer;

int fstatfs (FD, StatusBuffer)
int FD;
struct statfs *StatusBuffer;
```
int statfs64 (Path, StatusBuffer64)
char *Path;
struct statfs64 *StatusBuffer64;

int fstatfs64 (FileDescriptor, StatusBuffer64)
int FileDescriptor;
struct statfs64 *StatusBuffer64;
#include <sys/types.h>
#include <ustat.h>

int ustat (Device, Buffer)
dev_t Device;
struct ustat *Buffer;

Description
The `statfs` and `fstatfs` subroutines return information about the mounted file system that contains the file
named by the `Path` or `FileDescriptor` parameters. The returned information is in the format of a `statfs`
structure, described in the `sys/statfs.h` file.

The `statfs64` and `fstatfs64` subroutines are similar to the `statfs` and `fstatfs` subroutines except that the
returned information is in the format of a `statfs64` structure, described in the `sys/statfs.h` file, instead of a
`statfs` structure.

The `statfs64` structure provides invariant 64-bit fields for the file system blocks (or inodes) sizes or counts,
and the file system ID. This structure allows `statfs64` and `fstatfs64` to always return the specified
information in invariant 64-bit sizes.

The `ustat` subroutine also returns information about a mounted file system identified by `Device`. This
device identifier is for any given file and can be determined by examining the `st_dev` field of the structure
defined in the `sys/stat.h` file. The returned information is in the format of a `ustat` structure,
described in the `ustat.h` file. The `ustat` subroutine is superseded by the `statfs` and `fstatfs` subroutines.
Use one of these (`statfs` and `fstatfs`) subroutines instead.

Note: The `ustat` subroutine does not work for 64-bit sizes.

Parameters

Path
The path name of any file within the mounted file system.

FileDescriptor
A file descriptor obtained by a successful `open` or `fcntl` subroutine. A file descriptor is a
small positive integer used instead of a file name.

StatusBuffer
A pointer to a `statfs` buffer for the returned information from the `statfs` or `fstatfs`
subroutine.

StatusBuffer64
A pointer to a `statfs64` buffer for the returned information from the `statfs64` or `fstatfs64`
subroutine.

Device
The ID of the device. It corresponds to the `st_rdev` field of the structure returned by the
`stat` subroutine. The `stat` subroutine and the `sys/stat.h` file provide more information
about the device driver.

Buffer
A pointer to a `ustat` buffer to hold the returned information.

Return Values
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, and the `errno`
global variable is set to indicate the error.
Error Codes
The statfs, fstatfs, statfs64, fstatfs64, and ustat subroutines fail if the following is true:

EFAULT The Buffer parameter points to a location outside of the allocated address space of the process.

The fstatfs or fstatfs64 subroutine fails if the following is true:

EBADF The FileDescriptor parameter is not a valid file descriptor.
EIO An I/O error occurred while reading from the file system.

The statfs or statfs64 subroutine can be unsuccessful for other reasons. For a list of additional errors, see "Base Operating System Error Codes For Services That Require Path-Name Resolution".

Related Information
The stat ("statx, stat, lstat, fstat, fullstat, ffullstat, stat64, lstat64, fstat64, stat64x, fstat64x, or lstat64x Subroutine" on page 318) subroutine.

Files, Directories, and File Systems for Programmers in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

statvfs, fstatvfs, statvfs64, or fstatvfs64 Subroutine

Purpose
Returns information about a file system.

Library
Standard C Library (libc.a)

Syntax

```c
#include <sys/statvfs.h>

int statvfs (const char *Path, struct statvfs *Buf);

int fstatvfs (int Fildes, struct statvfs *Buf);

int statvfs64 (const char *Path, struct statvfs64 *Buf);

int fstatvfs64 (int Fildes, struct statvfs64 *Buf);
```

Description
The statvfs and fstatvfs subroutines return descriptive information about a mounted file system containing the file referenced by the Path or Fildes parameters. The Buf parameter is a pointer to a structure which will by filled by the subroutine call.
The *Path* and *Fildes* parameters must reference a file which resides on the file system. 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.

The *statvfs64* and *fstatvfs64* subroutines are similar to the *statvfs* and *fstatvfs* subroutines except that the returned information is in the format of a *statvfs64* structure instead of a *statvfs* structure.

The *statvfs64* structure provides invariant 64-bit fields for the file system blocks (or inodes) sizes and counts, and the file system ID. This structure allows *statvfs64* and *fstatvfs64* to always return the specified information in invariant 64-bit values.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>The path name identifying the file.</td>
</tr>
<tr>
<td>Buf</td>
<td>A pointer to a <em>statvfs</em> or <em>statvfs64</em> structure in which information is returned. The <em>statvfs</em> or <em>statvfs64</em> structure is described in the <em>sys/statvfs.h</em> header file.</td>
</tr>
<tr>
<td>Fildes</td>
<td>The file descriptor identifying the open file.</td>
</tr>
</tbody>
</table>

### Return Values

- 0  Successful completion.
- -1 Not successful and *errno* set to one of the following.

### Error Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EACCES</td>
<td>Search permission is denied on a component of the path.</td>
</tr>
<tr>
<td>EBADF</td>
<td>The file referred to by the <em>Fildes</em> parameter is not an open file descriptor.</td>
</tr>
<tr>
<td>EIO</td>
<td>An I/O error occurred while reading from the filesystem.</td>
</tr>
<tr>
<td>ELOOP</td>
<td>Too many symbolic links encountered in translating path.</td>
</tr>
<tr>
<td>ENAMETOOLONG</td>
<td>The length of the pathname exceeds <em>PATH_MAX</em>, or name component is longer than <em>NAME_MAX</em>.</td>
</tr>
<tr>
<td>ENOENT</td>
<td>The file referred to by the <em>Path</em> parameter does not exist.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>A memory allocation failed during information retrieval.</td>
</tr>
<tr>
<td>ENOTDIR</td>
<td>A component of the <em>Path</em> parameter prefix is not a directory.</td>
</tr>
<tr>
<td>EOVERFLOW</td>
<td>One of the values to be returned cannot be represented correctly in the structure pointed to by <em>buf</em>.</td>
</tr>
</tbody>
</table>

### Related Information

The *stat* ("*statx, stat, lstat, fstatx, fstat, fullstat, ffullstat, stat64, lstat64, fstat64, stat64x, fstat64x, or lstat64x Subroutine" subroutine), *statfs* ("*statfs, fstatfs, statfs64, fstatfs64, or ustat Subroutine" on page 315) subroutine.

### Purpose

Provides information about a file or shared memory object.
Library

Standard C Library (libc.a)

Syntax

#include <sys/stat.h>

int stat (Path, Buffer)
const char *Path;
struct stat *Buffer;
int lstat (Path, Buffer)
const char *Path;
struct stat *Buffer;

int fstat (FileDescriptor, Buffer)
int FileDescriptor;
struct stat *Buffer;

int statx (Path, Buffer, Length, Command)
char *Path;
struct stat *Buffer;
int lstatx (Path, Buffer, Length, Command)

#include <sys/fullstat.h>

int fullstat (Path, Command, Buffer)
struct fullstat *Buffer;
int ffullstat (FileDescriptor, Command, Buffer)
struct fullstat *Buffer;

int stat64 (Path, Buffer)
const char *Path;
struct stat64 *Buffer;
int lstat64 (Path, Buffer)
const char *Path;
struct stat64 *Buffer;

int fstat64 (FileDescriptor, Buffer)
int FileDescriptor;
struct stat64 *Buffer;

int stat64x (Path, Buffer)
const char *Path;
struct stat64x *Buffer;
int lstat64x (Path, Buffer)
const char *Path;
struct stat64x *Buffer;
int fstat64x (FileDescriptor, Buffer)
int FileDescriptor;
struct stat64x *Buffer;
Description
The **stat** subroutine obtains information about the file named by the **Path** parameter. Read, write, or execute permission for the named file is not required, but all directories listed in the path leading to the file must be searchable. The file information, which is a subset of the **stat** structure, is written to the area specified by the **Buffer** parameter.

The **lstat** subroutine obtains information about a file that is a symbolic link. The **lstat** subroutine returns information about the link, while the **stat** subroutine returns information about the file referenced by the link.

The **fstat** subroutine obtains information about the open file or shared memory object referenced by the **FileDescriptor** parameter. The **fstatx** subroutine obtains information about the open file or shared memory object referenced by the **FileDescriptor** parameter, as in the **fstat** subroutine.

The **st_mode**, **st_dev**, **st_uid**, **st_gid**, **st_atime**, **st_ctime**, and **st_mtime** fields of the **stat** structure have meaningful values for all file types. The **statx**, **stat**, **lstat**, **fstatx**, **fullstat**, or **ffullstat** subroutine sets the **st_nlink** field to a value equal to the number of links to the file.

The **statx** subroutine obtains a greater set of file information than the **stat** subroutine. The **Path** parameter is processed differently, depending on the contents of the **Command** parameter. The **Command** parameter provides the ability to collect information about symbolic links (as with the **lstat** subroutine) as well as information about mount points and hidden directories. The **statx** subroutine returns the amount of information specified by the **Length** parameter.

The **fullstat** and **ffullstat** subroutines are interfaces maintained for backward compatibility. With the exception of some field names, the **fullstat** structure is identical to the **stat** structure.

The **stat64**, **lstat64**, and **fstat64** subroutines are similar to the **stat**, **lstat**, and **fstat** subroutines except that they return file information in a **stat64** structure instead of a **stat** structure. The information is identical except that the **st_size** field is defined to be a 64-bit size. This allows **stat64**, **lstat64**, and **fstat64** to return file sizes which are greater than **OFF_MAX** (2 gigabytes minus 1).

In the large file enabled programming environment, **stat** is redefined to be **stat64**, **lstat** is redefined to be **lstat64**, and **fstat** is redefined to be **fstat64**.

The **stat64x**, **lstat64x**, and **fstat64x** subroutines are similar to the **stat**, **lstat**, and **fstat** subroutines except that they return file information in a **stat64x** structure instead of a **stat** structure. The information is identical except the following fields are defined to be 64-bit sizes: **st_dev**, **st_ino**, **st_rdev**, **st_size**, **st_atime**, **st_mtime**, **st_ctime**, **st_blksize**, and **st_blocks**.

**Note:** The 64-bit **st_dev** field always contains a 64-bit device ID, where the first two bits are reserved, the next 30 bits are the device major number, and the next 32 bits are the device minor number.

This allows **stat64x**, **lstat64x**, and **fstat64x** to return the specified information in invariant 64-bit sizes, regardless of the mode of an application or the kernel it is running on.

Parameters
**Path** Specifies the path name identifying the file. This name is interpreted differently depending on the interface used.

**FileDescriptor** Specifies the file descriptor identifying the open file or shared memory object.
Note: If the FileDescriptor parameter references a shared memory object, only the st_uid, st_gid, st_size, and st_mode fields of the stat structure are filled, and only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits are valid.

Buffer Specifies a pointer to the stat structure in which information is returned. The stat structure is described in the sys/stat.h file.

Length Indicates the amount of information, in bytes, to be returned. Any value between 0 and the value returned by the STATXSIZE macro, inclusive, may be specified. The following macros may be used:

STATSIZE Specifies the subset of the stat structure that is normally returned for a stat call.

FULLSTATSIZE Specifies the subset of the stat (fullstat) structure that is normally returned for a fullstat call.

STATXSIZE Specifies the complete stat structure. 0 specifies the complete stat structure, as if STATXSIZE had been specified.

Command Specifies a processing option. For the statx subroutine, the Command parameter determines how to interpret the path name provided, specifically, whether to retrieve information about a symbolic link, hidden directory, or mount point. Flags can be combined by logically ORing them together. The following options are possible values:

STX_LINK If the Command parameter specifies the STX_LINK flag and the Path parameter is a path name that refers to a symbolic link, the statx subroutine returns information about the symbolic link. If the STX_LINK flag is not specified, the statx subroutine returns information about the file to which the link refers.

If the Command parameter specifies the STX_LINK flag and the Path value refers to a symbolic link, the st_mode field of the returned stat structure indicates that the file is a symbolic link.

STX_HIDDEN If the Command parameter specifies the STX_HIDDEN flag and the Path value is a path name that refers to a hidden directory, the statx subroutine returns information about the hidden directory. If the STX_HIDDEN flag is not specified, the statx subroutine returns information about a subdirectory of the hidden directory.

If the Command parameter specifies the STX_HIDDEN flag and Path refers to a hidden directory, the st_mode field of the returned stat structure indicates that this is a hidden directory.

STX_MOUNT If the Command parameter specifies the STX_MOUNT flag and the Path value is the name of a file or directory that has been mounted over, the statx subroutine returns information about the mounted-over file. If the STX_MOUNT flag is not specified, the statx subroutine returns information about the mounted file or directory (the root directory of a virtual file system).

If the Command parameter specifies the STX_MOUNT flag, the FS_MOUNT bit in the st_flag field of the returned stat structure is set if, and only if, this file is mounted over.
If the Command parameter does not specify the STX_MOUNT flag, the FS_MOUNT bit in the s_flag field of the returned stat structure is set if, and only if, this file is the root directory of a virtual file system.

**STX_NORMAL**

If the Command parameter specifies the STX_NORMAL flag, then no special processing is performed on the Path value. This option should be used when STX_LINK, STX_HIDDEN, and STX_MOUNT flags are not desired.

For the fstatx subroutine, there are currently no special processing options. The only valid value for the Command parameter is the STX_NORMAL flag.

For the fullstat and ffullstat subroutines, the Command parameter may specify the FL_STAT flag, which is equivalent to the STX_NORMAL flag, or the FL_NOFOLLOW flag, which is equivalent to STX_LINK flag.

**STX_64**

If the Command parameter specifies the STX_64 flag and the file size is greater than OFF_MAX, then statx succeeds and returns the file size. Otherwise, statx fails and sets the errno to EOVERFLOW.

**STX_64X**

If the Command parameter specifies the STX_64X flag and the stat structure size is not equal to the size of STX_64X, statx fails and sets the errno to EINVAL.

**Return Values**

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the errno global variable is set to indicate the error.

**Error Codes**

The stat, lstat, statx, and fullstat subroutines are unsuccessful if one or more of the following are true:

- **EACCES** Search permission is denied for one component of the path prefix.
- **ENAMETOOLONG** The length of the path prefix exceeds the PATH_MAX flag value or a path name is longer than the NAME_MAX flag value while the POSIX_NO_TRUNC flag is in effect.
- **ENOTDIR** A component of the path prefix is not a directory.
- **EFAULT** Either the Path or the Buffer parameter points to a location outside of the allocated address space of the process.
- **ENOENT** The file named by the Path parameter does not exist.
- **EOVERFLOW** The size of the file is larger than can be represented in the stat structure pointed to by the Buffer parameter.

The stat, lstat, statx, and fullstat subroutines can be unsuccessful for other reasons. See "Base Operating System Error Codes for Services that Require Path-Name Resolution" for a list of additional errors.

The fstat, fstatx, and ffullstat subroutines fail if one or more of the following are true:

- **EBADF** The FileDescriptor parameter is not a valid file descriptor.
- **EFAULT** The Buffer parameter points to a location outside the allocated address space of the process.
- **EIO** An input/output (I/O) error occurred while reading from the file system.

The statx and fstatx subroutines are unsuccessful if one or more of the following are true:

- **EINVAL** The Length value is not between 0 and the value returned by the STATSIZE macro, inclusive.
- **EINVAL** The Command parameter contains an unacceptable value.
Related Information
The chmod subroutine, chown subroutine, link subroutine, mknod subroutine, mount subroutine, open subroutine, openx subroutine, open, or creat subroutine, pipe subroutine, symlink subroutine, symlink subroutine, vtimes subroutine.


strcat, strncat, strxfrm, strcpy, strncpy, or strdup Subroutine

Purpose
Copies and appends strings in memory.

Library
Standard C Library (libc.a)

Syntax

#include <string.h>

char *strcat (String1, String2)
char *String1;
const char *String2;

char *strncat (String1, String2, Number)
char *String1;
const char *String2;
size_t Number;

size_t strxfrm (String1, String2, Number)
char *String1;
const char *String2;
size_t Number;

char *strcpy (String1, String2)
char *String1;
const char *String2;

char *strncpy (String1, String2, Number)
char *String1;
const char *String2;
size_t Number;

char *strdup (String1)
const char *String1;
Description

The `strcat`, `strncat`, `strxfrm`, `strcpy`, `strncpy`, and `strdup` subroutines copy and append strings in memory.

The `String1` and `String2` parameters point to strings. A string is an array of characters terminated by a null character. The `strcat`, `strncat`, `strcpy`, and `strncpy` subroutines all alter the string in the `String1` parameter. However, they do not check for overflow of the array to which the `String1` parameter points. String movement is performed on a character-by-character basis and starts at the left. Overlapping moves toward the left work as expected, but overlapping moves to the right may give unexpected results. All of these subroutines are declared in the `string.h` file.

The `strcat` subroutine adds a copy of the string pointed to by the `String2` parameter to the end of the string pointed to by the `String1` parameter. The `strcat` subroutine returns a pointer to the null-terminated result.

The `strncat` subroutine copies a number of bytes specified by the `Number` parameter from the `String2` parameter to the end of the string pointed to by the `String1` parameter. The subroutine stops copying before the end of the number of bytes specified by the `Number` parameter if it encounters a null character in the `String2` parameter’s string. The `strcat` subroutine returns a pointer to the null-terminated result. The `strncat` subroutine returns the value of the `String1` parameter.

The `strxfrm` subroutine transforms the string pointed to by the `String2` parameter and places it in the array pointed to by the `String1` parameter. The `strxfrm` subroutine transforms the entire string if possible, but places no more than the number of bytes specified by the `Number` parameter in the array pointed to by the `String1` parameter. Consequently, if the `Number` parameter has a value of 0, the `String1` parameter can be a null pointer. The `strxfrm` subroutine returns the length of the transformed string, not including the terminating null byte. If the returned value is equal to or more than that of the `Number` parameter, the contents of the array pointed to by the `String1` parameter are indeterminable. If the number of bytes specified by the `Number` parameter is 0, the `strxfrm` subroutine returns the length required to store the transformed string, not including the terminating null byte. The `strxfrm` subroutine is determined by the `LC_COLLATE` category.

The `strcpy` subroutine copies the string pointed to by the `String2` parameter to the character array pointed to by the `String1` parameter. Copying stops after the null character is copied. The `strcpy` subroutine returns the value of the `String1` parameter, if successful. Otherwise, a null pointer is returned.

The `strncpy` subroutine copies the number of bytes specified by the `Number` parameter from the string pointed to by the `String2` parameter to the character array pointed to by the `String1` parameter. If the `String2` parameter value is less than the specified number of characters, then the `strncpy` subroutine pads the `String1` parameter with trailing null characters to a number of bytes equaling the value of the `Number` parameter. If the `String2` parameter is exactly the specified number of characters or more, then only the number of characters specified by the `Number` parameter are copied and the result is not terminated with a null byte. The `strncpy` subroutine returns the value of the `String1` parameter.

The `strdup` subroutine returns a pointer to a new string, which is a duplicate of the string pointed to by the `String1` parameter. Space for the new string is obtained by using the `malloc` subroutine. A null pointer is returned if the new string cannot be created.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Number</code></td>
<td>Specifies the number of bytes in a string to be copied or transformed.</td>
</tr>
<tr>
<td><code>String1</code></td>
<td>Points to a string to which the specified data is copied or appended.</td>
</tr>
<tr>
<td><code>String2</code></td>
<td>Points to a string which contains the data to be copied, appended, or transformed.</td>
</tr>
</tbody>
</table>
Error Codes
The `strcat`, `strncat`, `strxfrm`, `strcpyp`, `strncpy`, and `strdup` subroutines fail if the following occurs:

**EFAULT**  A string parameter is an invalid address.

In addition, the `strxfrm` subroutine fails if:

**EINVAL**  A string parameter contains characters outside the domain of the collating sequence.

Related Information
The `memccpy`, `memchr`, `memcmp`, `memcpy`, or `memmove` subroutine, `setlocale` subroutine, `strcmp`, `strncm`, `strccasecmp`, `strncasecmp`, or `strcoll` subroutine, `strcspn`, `strstr`, or `strtok` subroutine on page 333 subroutine, `swab` subroutine.

Subroutines, Example Programs, and Libraries and List of String Manipulation Services in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.


### strcmp, strncmp, strccasecmp, strncasecmp, or strcoll Subroutine

**Purpose**

Compares strings in memory.

**Library**

Standard C Library (`libc.a`)

**Syntax**

```c
#include <string.h>

int strcmp (const char *String1, *String2);

int strncmp (const char *String1, *String2, size_t Number);

int strcoll (const char *String1, *String2);

#include <strings.h>

int strcasecmp (const char *String1, *String2);
```
Description

The `strcmp`, `strcasecmp`, `strncasecmp`, and `strcoll` subroutines compare strings in memory.

The `String1` and `String2` parameters point to strings. A string is an array of characters terminated by a null character.

The `strcmp` subroutine performs a case-sensitive comparison of the string pointed to by the `String1` parameter and the string pointed to by the `String2` parameter, and analyzes the extended ASCII character set values of the characters in each string. The `strcmp` subroutine compares `unsigned char` data types. The `strcmp` subroutine then returns a value that is:

- Less than 0 if the value of string `String1` is lexicographically less than string `String2`.
- Equal to 0 if the value of string `String1` is lexicographically equal to string `String2`.
- Greater than 0 if the value of string `String1` is lexicographically greater than string `String2`.

The `strncmp` subroutine makes the same comparison as the `strcmp` subroutine, but compares up to the maximum number of pairs of bytes specified by the `Number` parameter.

The `strcasecmp` subroutine performs a character-by-character comparison similar to the `strcmp` subroutine. However, the `strcasecmp` subroutine is not case-sensitive. Uppercase and lowercase letters are mapped to the same character set value. The sum of the mapped character set values of each string is used to return a value that is:

- Less than 0 if the value of string `String1` is lexicographically less than string `String2`.
- Equal to 0 if the value of string `String1` is lexicographically equal to string `String2`.
- Greater than 0 if the value of string `String1` is lexicographically greater than string `String2`.

The `strncasecmp` subroutine makes the same comparison as the `strcasecmp` subroutine, but compares up to the maximum number of pairs of bytes specified by the `Number` parameter.

Note: Both the `strcasecmp` and `strncasecmp` subroutines only work with 7-bit ASCII characters.

The `strcoll` subroutine works the same as the `strcmp` subroutine, except that the comparison is based on a collating sequence determined by the `LC_COLLATE` category. If the `strcmp` subroutine is used on transformed strings, it returns the same result as the `strcoll` subroutine for the corresponding untransformed strings.

Parameters

- `Number` The number of bytes in a string to be examined.
- `String1` Points to a string which is compared.
- `String2` Points to a string which serves as the source for comparison.

Error Codes

The `strcmp`, `strncmp`, `strcasecmp`, `strncasecmp`, and `strcoll` subroutines fail if the following occurs:

- `EFAULT` A string parameter is an invalid address.
In addition, the `strcoll` subroutine fails if:

**EINVAL** A string parameter contains characters outside the domain of the collating sequence.

### Related Information

The `memccpy`, `memchr`, `memcmp`, `memcpy`, or `memmove` subroutine, `setlocale` subroutine, `strcat`, `strncat`, `strxfrm`, `strcpy`, or `strdup` subroutine, `strncpy` subroutine, `strlen`, `strchr`, `strrchr`, `strpbrk`, `strspn`, `strcspn`, `strstr`, or `strtok` subroutine, `swab` subroutine.

List of String Manipulation Subroutines and Subroutines, Example Programs, and Libraries in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.


---

### `strerror` Subroutine

#### Purpose
Maps an error number to an error message string.

#### Library
Standard C Library (`libc.a`)

#### Syntax
```
#include <string.h>

char *strerror (ErrorNumber)
    int ErrorNumber;
```

#### Description

**Attention:** Do not use the `strerror` subroutine in a multithreaded environment.

The `strerror` subroutine maps the error number in the `ErrorNumber` parameter to the error message string. The `strerror` subroutine retrieves an error message based on the current value of the `LC_MESSAGES` category. If the specified message catalog cannot be opened, the default message is returned. The returned message does not contain a new line ("\n").

#### Parameters

- **`ErrorNumber`** Specifies the error number to be associated with the error message.

#### Return Values

The `strerror` subroutine returns a pointer to the error message.
strfmon Subroutine

Purpose
Formats monetary strings.

Library
Standard C Library (libc.a)

Syntax
#include <monetary.h>

ssize_t strfmon (S, MaxSize, Format, ...)
char *S;
size_t MaxSize;
const char *Format, ...;

Description
The strfmon subroutine converts numeric values to monetary strings according to the specifications in the Format parameter. This parameter also contains numeric values to be converted. Characters are placed into the S array, as controlled by the Format parameter. The LC_MONETARY category governs the format of the conversion.

The strfmon subroutine can be called multiple times by including additional format structures, as specified by the Format parameter.

The Format parameter specifies a character string that can contain plain characters and conversion specifications. Plain characters are copied to the output stream. Conversion specifications result in the fetching of zero or more arguments, which are converted and formatted.

If there are insufficient arguments for the Format parameter, the results are undefined. If arguments remain after the Format parameter is exhausted, the excess arguments are ignored.

A conversion specification consists of the following items in the following order: a % (percent sign), optional flags, optional field width, optional left precision, optional right precision, and a required conversion character that determines the conversion to be performed.

Parameters

S Contains the output of the strfmon subroutine.
MaxSize Specifies the maximum number of bytes (including the null terminating byte) that may be placed in the S parameter.
Format Contains characters and conversion specifications.
Flags
One or more of the following flags can be specified to control the conversion:

= f  An = (equal sign) followed by a single character that specifies the numeric fill character. 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 is specified.

^  Does not use grouping characters when formatting the currency amount. The default is to insert grouping characters if defined for the current locale.

+ or ( Determines the representation of positive and negative currency amounts. Only one of these flags may be specified. The locale’s equivalent of + (plus sign) and - (negative sign) are used if + is specified. The locale’s equivalent of enclosing negative amounts within parentheses is used if ( (left parenthesis) is specified. If neither flag is included, a default specified by the current locale is used.

-  Left-justifies all fields (pads to the right). The default is right-justification.

!  Suppresses the currency symbol from the output conversion.

Field Width

w  The decimal-digit string w specifies the minimum field width in which the result of the conversion is right-justified. If -w is specified, the result is left-justified. The default is a value of 0.

Left Precision

# n  A # (pound sign) followed by a decimal-digit string, n, specifies the maximum number of digits to be formatted to the left of the radix character. This option can be specified to keep formatted output from multiple calls to the strfmt subroutine aligned in the same columns. It can also be used to fill unused positions with a special character (for example, $***123.45). This option causes an amount to be formatted as if it has the number of digits specified by the n variable. If more than n digit positions are required, this option is ignored. Digit positions in excess of those required are filled with the numeric fill character set with the =f flag.

If defined for the current locale and not suppressed with the ^ flag, the subroutine inserts grouping characters before fill characters (if any). Grouping characters are not applied to fill characters, even if the fill character is a digit. In the example:

$000001,234.56

grouping characters do not appear after the first or fourth 0 from the left.

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 equal in length.

Right Precision

.p  A . (period) followed by a decimal digit string, p, specifies the number of digits after the radix character. If the value of the p variable is 0, no radix character is used. If a right precision is not specified, a default specified by the current locale is use. The amount being formatted is rounded to the specified number of digits prior to formatting.

Conversion Characters

i  The double argument is formatted according to the current locale’s international currency format; for example, in the U.S.: 1,234.56.

n  The double argument is formatted according to the current locale’s national currency format; for example, in the U.S.: $1,234.56.

%  No argument is converted; the conversion specification %% is replaced by a single %.
Return Values
If successful, and if the number of resulting bytes (including the terminating null character) is not more than the number of bytes specified by the MaxSize parameter, the strfmon subroutine returns the number of bytes placed into the array pointed to by the S parameter (not including the terminating null byte). Otherwise, a value of -1 is returned and the contents of the S array are indeterminate.

Error Codes
The strfmon subroutine may fail if the following is true:
E2BIG Conversion stopped due to lack of space in the buffer.

Related Information
The scanf, fscanf, sscanf, or wsscanf subroutine, strftime subroutine, strptime strftime Subroutine on page 343 subroutine, wcsftime wcsftime Subroutine on page 496 subroutine.

Subroutines, Example Programs, and Libraries in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.


strftime Subroutine

Purpose
Formats time and date.

Library
Standard C Library (libc.a)

Syntax
#include <time.h>

size_t strftime (String, Length, Format, TmDate)
char *String;
size_t Length;
const char *Format;
const struct tm *TmDate;

Description
The strftime subroutine converts the internal time and date specification of the tm structure, which is pointed to by the TmDate parameter, into a character string pointed to by the String parameter under the direction of the format string pointed to by the Format parameter. The actual values for the format specifiers are dependent on the current settings for the LC_TIME category. The tm structure values may be assigned by the user or generated by the localtime or gmtime subroutine. The resulting string is similar to the result of the printf Format parameter, and is placed in the memory location addressed by the String parameter. The maximum length of the string is determined by the Length parameter and terminates with a null character.

Many conversion specifications are the same as those used by the date command. The interpretation of some conversion specifications is dependent on the current locale of the process.
The *Format* parameter is a character string containing two types of objects: plain characters that are simply placed in the output string, and conversion specifications that convert information from the *TmDate* parameter into readable form in the output string. Each conversion specification is a sequence of this form:

% type

- A % (percent sign) introduces a conversion specification.
- The type of conversion is specified by one or two conversion characters. The characters and their meanings are:

- %a Represents the locale’s abbreviated weekday name (for example, Sun) defined by the *abday* statement in the LC_TIME category.
- %A Represents the locale’s full weekday name (for example, Sunday) defined by the *day* statement in the LC_TIME category.
- %b Represents the locale’s abbreviated month name (for example, Jan) defined by the *abmon* statement in the LC_TIME category.
- %B Represents the locale’s full month name (for example, January) defined by the *mon* statement in the LC_TIME category.
- %c Represents the locale’s date and time format defined by the *d_t_fmt* statement in the LC_TIME category.
- %C Represents the century number (the year divided by 100 and truncated to an integer) as a decimal number (00 through 99).
- %d Represents the day of the month as a decimal number (01 to 31).
- %D Represents the date in %m/%d/%y format (for example, 01/31/91).
- %e Represents the day of the month as a decimal number (01 to 31). The %e field descriptor uses a two-digit field. If the day of the month is not a two-digit number, the leading digit is filled with a space character.
- %E Represents the locale’s combined alternate era year and name, respectively, in %o %N format.
- %h Represents the locale’s abbreviated month name (for example, Jan) defined by the *abmon* statement in the LC_TIME category. This field descriptor is a synonym for the %b field descriptor.
- %H Represents the 24-hour-clock hour as a decimal number (00 to 23).
- %I Represents the 12-hour-clock hour as a decimal number (01 to 12).
- %j Represents the day of the year as a decimal number (001 to 366).
- %k Represents the 24-hour-clock hour as a right-justified space-filled number (0 to 23).
- %m Represents the month of the year as a decimal number (01 to 12).
- %M Represents the minutes of the hour as a decimal number (00 to 59).
- %n Specifies a new-line character.
- %N Represents the locale’s alternate era name.
- %o Represents the alternate era year.
- %p Represents the locale’s a.m. or p.m. string defined by the *am_pm* statement in the LC_TIME category.
- %r Represents 12-hour clock time with a.m./p.m. notation as defined by the *t_fmt_ampm* statement. The usual format is %I:%M:%S %p.
- %R Represents 24-hour clock time in %H:%M format.
- %s Represents the number of seconds since January 1, 1970, Coordinated Universal Time (CUT).
- %S Represents the seconds of the minute as a decimal number (00 to 59).
- %t Specifies a tab character.
- %T Represents 24-hour-clock time in the format %H:%M:%S (for example, 16:55:15).
- %u Represents the weekday as a decimal number (1 to 7). Monday or its equivalent is considered the first day of the week for calculating the value of this field descriptor.
- %U Represents the week of the year as a decimal number (00 to 53). Sunday, or its equivalent as defined by the *day* statement in the LC_TIME category, is considered the first day of the week for calculating the value of this field descriptor.
- %V Represents the week number of the year (with Monday as the first day of the week) as a decimal number (01 to 53). If the week containing January 1 has four or more days in the new year, then it is considered week 1; otherwise, it is considered week 53 of the previous year, and the next week is week 1 of the new year.
- %w Represents the day of the week as a decimal number (0 to 6). Sunday, or its equivalent as defined by the *day* statement, is considered as 0 for calculating the value of this field descriptor.
- %W Represents the week of the year as a decimal number (00 to 53). Monday, or its equivalent as defined by the *day* statement, is considered the first day of the week for calculating the value of this field descriptor.
- %x Represents the locale’s date format as defined by the *d_fmt* statement.
%X Represents the locale's time format as defined by the t_fmt statement.
%y Represents the year of the century.
Note: When the environment variable XPG_TIME_FMT=ON, %y is the year within the century. When a century is not otherwise specified, values in the range 69-99 refer to years in the twentieth century (1969 to 1999, inclusive); values in the range 00-68 refer to 2000 to 2068, inclusive.
%Y Represents the year as a decimal number (for example, 1989).
%Z Represents the time-zone name if one can be determined (for example, EST). No characters are displayed if a time zone cannot be determined.
%% Specifies a % (percent sign).

Some conversion specifiers can be modified by the E or O modifier characters to indicate that an alternative format or specification should be used. If the alternative format or specification does not exist for the current locale, the behavior will be the same as with the unmodified conversion specification. The following modified conversion specifiers are supported:

%Ec Represents the locale's alternative appropriate date and time as defined by the era_d_t_fmt statement.
%EC Represents the name of the base year (or other time period) in the locale's alternative form as defined by the era statement under the era_name category of the current era.
%Ex Represents the locale's alternative date as defined by the era_d_fmt statement.
%EX Represents the locale's alternative time as defined by the era_t_fmt statement.
%Ey Represents the offset from the %EC modified conversion specifier (year only) in the locale's alternative form.
%EY Represents the full alternative-year form.
%Od Represents the day of the month, using the locale's alternative numeric symbols, filled as needed with leading 0’s if an alternative symbol for 0 exists. If an alternative symbol for 0 does not exist, the %Od modified conversion specifier uses leading space characters.
%Oe Represents the day of the month, using the locale's alternative numeric symbols, filled as needed with leading 0’s if an alternative symbol for 0 exists. If an alternative symbol for 0 does not exist, the %Oe modified conversion specifier uses leading space characters.
%OH Represents the hour in 24-hour clock time, using the locale’s alternative numeric symbols.
%OI Represents the hour in 12-hour clock time, using the locale’s alternative numeric symbols.
%Om Represents the month, using the locale’s alternative numeric symbols.
%OM Represents the minutes, using the locale’s alternative numeric symbols.
%Os Represents the seconds, using the locale’s alternative numeric symbols.
%Ou Represents the weekday as a number using the locale's alternative numeric symbols.
%OU Represents the week number of the year, using the locale’s alternative numeric symbols. Sunday is considered the first day of the week. Use the rules corresponding to the %U conversion specifier.
%OV Represents the week number of the year (Monday as the first day of the week, rules corresponding to %V) using the locale’s alternative numeric symbols.
%Ow Represents the number of the weekday (with Sunday equal to 0), using the locale’s alternative numeric symbols.
%OW Represents the week number of the year using the locale’s alternative numeric symbols. Monday is considered the first day of the week. Use the rules corresponding to the %W conversion specifier.
%Oy Represents the year (offset from %C) using the locale’s alternative numeric symbols.

Parameters

String Points to the string to hold the formatted time.
Length Specifies the maximum length of the string pointed to by the String parameter.
Format Points to the format character string.
TmDate Points to the time structure that is to be converted.
Return Values
If the total number of resulting bytes, including the terminating null byte, is not more than the Length value, the strftime subroutine returns the number of bytes placed into the array pointed to by the String parameter, not including the terminating null byte. Otherwise, a value of 0 is returned and the contents of the array are indeterminate.

Related Information
The localtime subroutine, gmtime subroutine, mbstowcs subroutine, printf subroutine, strfmon subroutine, strftime subroutine, strftime Subroutine on page 328 subroutine, strftime (“strftime Subroutine” on page 343) subroutine, wcsftime subroutine.

The date command.

LC_TIME Category for the Locale Definition Source File Format in AIX 5L Version 5.3 Files Reference.


, Subroutines, Example Programs, and Libraries in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.


strlen, strchr, strrchr, strpbrk, strspn, strcspn, strstr, strtok, or strsep Subroutine

Purpose
Determines the size, location, and existence of strings in memory.

Library
Standard C Library (libc.a)

Syntax

```c
#include <string.h>

size_t strlen (String)
const char *String;

cchar *strchr (String, Character)
const char *String;
int Character;

cchar *strchr (String, Character)
const char *String;
int Character;

cchar *strpbrk (String1, String2)
const char *String1, * String2;

size_t strspn (String1, String2)
const char *String1, * String2;

size_t strcspn (String1, String2)
const char *String1, *String2;
```
Description

Attention: Do not use the `strtok` subroutine in a multithreaded environment. Use the `strtok_r` subroutine instead.

The `strlen`, `strchr`, `strchr`, `strpbrk`, `strspn`, `strcspn`, `strstr`, and `strtok` subroutines determine such values as size, location, and the existence of strings in memory.

The `String1`, `String2`, and `String` parameters point to strings. A string is an array of characters terminated by a null character.

The `strlen` subroutine returns the number of bytes in the string pointed to by the `String` parameter, not including the terminating null bytes.

The `strchr` subroutine returns a pointer to the first occurrence of the character specified by the `Character` (converted to an unsigned character) parameter in the string pointed to by the `String` parameter. A null pointer is returned if the character does not occur in the string. The null byte that terminates a string is considered to be part of the string.

The `strchr` subroutine returns a pointer to the last occurrence of the character specified by the `Character` (converted to a character) parameter in the string pointed to by the `String` parameter. A null pointer is returned if the character does not occur in the string. The null byte that terminates a string is considered to be part of the string.

The `strpbrk` subroutine returns a pointer to the first occurrence in the string pointed to by the `String1` parameter of any bytes from the string pointed to by the `String2` parameter. A null pointer is returned if no bytes match.

The `strspn` subroutine returns the length of the initial segment of the string pointed to by the `String1` parameter, which consists entirely of bytes from the string pointed to by the `String2` parameter.

The `strcspn` subroutine returns the length of the initial segment of the string pointed to by the `String1` parameter, which consists entirely of bytes not from the string pointed to by the `String2` parameter.

The `strstr` subroutine finds the first occurrence in the string pointed to by the `String1` parameter of the sequence of bytes specified by the string pointed to by the `String2` parameter (excluding the terminating null character). It returns a pointer to the string found in the `String1` parameter, or a null pointer if the string was not found. If the `String2` parameter points to a string of 0 length, the `strstr` subroutine returns the value of the `String1` parameter.
The `strtok` subroutine breaks the string pointed to by the `String1` parameter into a sequence of tokens, each of which is delimited by a byte from the string pointed to by the `String2` parameter. The first call in the sequence takes the `String1` parameter as its first argument and is followed by calls that take a null pointer as their first argument. The separator string pointed to by the `String2` parameter may be different from call to call.

The first call in the sequence searches the `String1` parameter for the first byte that is not contained in the current separator string pointed to by the `String2` parameter. If no such byte is found, no tokens exist in the string pointed to by the `String1` parameter, and a null pointer is returned. If such a byte is found, it is the start of the first token.

The `strtok` subroutine then searches from the first token 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 the `String1` parameter, and subsequent searches for a token return a null pointer. If such a byte is found, the `strtok` subroutine overwrites it with a null byte, which terminates the current token. The `strtok` subroutine saves a pointer to the following byte, from which the next search for a token will start. The subroutine returns a pointer to the first byte of the token.

Each subsequent call with a null pointer as the value of the first argument starts searching from the saved pointer, using it as the first token. Otherwise, the subroutine’s behavior does not change.

The `strsep` subroutine returns the next token from the string `String1` which is delimited by `String2`. The token is terminated with a `\0` character and `String1` is updated to point past the token. The `strsep` subroutine returns a pointer to the token, or NULL if `String2` is not found in `String1`.

The `index`, `rindex` and `strsep` subroutines are included for compatibility with BSD and are not part of the ANSI C Library. The `index` subroutine is implemented as a call to the `strchr` subroutine. The `rindex` subroutine is implemented as a call to the `strrchr` subroutine.

**Parameters**

- **Character**: Specifies a character for which to return a pointer.
- **String**: Points to a string from which data is returned.
- **String1**: Points to a string from which an operation returns results.
- **String2**: Points to a string which contains source for an operation.

**Error Codes**

The `strlen`, `strchr`, `strchr`, `strpbrk`, `strspn`, `strcspn`, `strstr`, and `strtok` subroutines fail if the following occurs:

- **EFAULT**: A string parameter is an invalid address.

**Related Information**

The `strcat, strncat, strxfrm, strcpy, strncpy, or strdup Subroutine` on page 323, `strcmp, strncmp, strcasecmp, strncasecmp, or strcoll Subroutine` on page 325, `strtok_r Subroutine` on page 340, and `swab Subroutine` on page 346.

The `memccpy, memchr, memcmp, memcmp, or memmove` subroutine in AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 1

List of String Manipulation Services and Subroutines, Example Programs, and Libraries in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
strncollen Subroutine

Purpose
Returns the number of collation values for a given string.

Library
Standard C Library (libc.a)

Syntax

```c
#include <string.h>

int strncollen (const char *String, const int Number);
```

Description
The `strncollen` subroutine returns the number of collation values for a given string pointed to by the `String` parameter. The count of collation values is terminated when either a null character is encountered or when the number of bytes indicated by the `Number` parameter have been examined.

The collation values are set by the `setlocale` subroutine for the `LC_COLLATE` category. For example, if the locale is set to Es_ES (Spanish spoken in Spain) for the `LC_COLLATE` category, where `ch` has one collation value, then `strncollen` ('abchd', 5) returns 4.

In German, the `<Sharp-S>` character has two collation values, so substituting the `<Sharp-S>` character for B in the following example, `strncollen` ('straBa', 6) returns 7.

If a character has no collation value, its collation length is 0.

Parameters

- `Number` The number of bytes in a string to be examined.
- `String` Pointer to a string to be examined for collation value.

Return Values
Upon successful completion, the `strncollen` subroutine returns the collation value for a given string, pointed to by the `String` parameter.

Related Information
The `setlocale` subroutine, `strcat`, `strncat`, `strxfrm`, `strcpy`, `strncpy`, or `strdup` subroutine, `strcmp`, `strncasecmp`, or `strcoll` subroutine.

strtof, strtod, or strtold Subroutine

Purpose
Converts a string to a double-precision number.

Syntax
#include <stdlib.h>

float strtof (nptr, endptr)
const char *restrict nptr;
char **restrict endptr;

double strtod (nptr, endptr)
const char *nptr
char**endptr;

long double strtold (nptr, endptr)
const char *restrict nptr;
char **restrict endptr;

Description
The strtof, strtod, and strtold subroutines convert the initial portion of the string pointed to by nptr to double, float, and long double representation, respectively. First, they decompose 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 or representing infinity or NaN.
- A final string of one or more unrecognized characters, including the terminating null byte of the input string.

Then, they attempt to convert the subject sequence to a floating-point number, and return the result.

The expected form of the subject sequence is an optional plus or minus sign, and one of the following:

- A non-empty sequence of decimal digits optionally containing a radix character, and an optional exponent part
- A 0x or 0X, and a non-empty sequence of hexadecimal digits optionally containing a radix character, and an optional binary exponent part
- One of INF or INFINITY, ignoring case
- One of NAN or NAN(n-char-sequence_opt), ignoring case in the NAN part, where:

n-char-sequence:
digit
nondigit
n-char-sequence digit
n-char-sequence nondigit

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 not of the expected form.

If the subject sequence has the expected form for a floating-point number, the sequence of characters starting with the first digit or the decimal-point character (whichever occurs first) are interpreted as a floating constant of the C language, except that the radix character is used in place of a period, and if neither an exponent part nor a radix character appears in a decimal floating-point number, or if a binary exponent part does not appear in a hexadecimal floating-point number, an exponent part of the appropriate type with value zero is assumed to follow the last digit in the string.
If the subject sequence begins with a minus sign, the sequence is interpreted as negated. A character sequence INF or INFINITY shall be interpreted as an infinity, if representable in the return type, or else as if it were a floating constant that is too large for the range of the return type. A character sequence NAN or NAN(n-char-sequence opt ) is interpreted as a quiet NaN, if supported in the return type, or else as if it were a subject sequence part that does not have the expected form. The meaning of the n-char sequences is implementation-defined. A pointer to the final string is stored in the object pointed to by the endptr parameter, provided that the endptr parameter is not a null pointer.

If the subject sequence has the hexadecimal form, the value resulting from the conversion is correctly rounded.

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 C or POSIX locales, other implementation-defined subject sequences may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion shall be performed; the value of str is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

The strtod subroutine does not change the setting of the errno global variable if successful.

Since 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, call the strtof or strtold subroutine, then check errno.

Parameters

nptr Specifies the string to be converted.
endptr Points to the final string.

Return Values

Upon successful completion, the strtof and strtold subroutines return the converted value. If no conversion could be performed, 0 is returned, and the errno global variable may be set to EINVAL.

If the correct value is outside the range of representable values, HUGE_VAL, HUGE_VALF, or HUGE_VALL is returned (according to the sign of the value), and errno is set to ERANGE.

If the correct value would cause an underflow, a value whose magnitude is no greater than the smallest normalized positive number in the return type is returned and the errno global variable is set to ERANGE.

Error Codes

Note: Because a value of 0 can indicate either an error or a valid result, an application that checks for errors with the strtod, strtof, and strtold subroutines should set the errno global variable equal to 0 prior to the subroutine call. The application can check the errno global variable after the subroutine call.

If the string pointed to by NumberPointer is empty or begins with an unrecognized character, a value of 0 is returned for the strtod, strtof, and strtold subroutines.

If the conversion cannot be performed, a value of 0 is returned, and the errno global variable is set to indicate the error.
If the conversion causes an overflow (that is, the value is outside the range of representable values), +/-HUGE_VAL is returned with the sign indicating the direction of the overflow, and the errno global variable is set to ERANGE.

If the conversion would cause an underflow, a properly signed value of 0 is returned and the errno global variable is set to ERANGE.

For the strtod, strtof, and strtol subroutines, if the value of the EndPointer parameter is not (char**) NULL, a pointer to the character that stopped the subroutine is stored in *EndPointer. If a floating-point value cannot be formed, *EndPointer is set to NumberPointer.

The strtof subroutine has only one rounding error. (If the strtod subroutine is used to create a double-precision floating-point number and then that double-precision number is converted to a floating-point number, two rounding errors could occur.)

Related Information
"scanf, fscanf, sscanf, or wsscanf Subroutine" on page 126, "setlocale Subroutine" on page 171, and "strtol, strtoul, strtoll, strtoull, or atoi Subroutine" on page 341.

cctype, isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, isgraph, iscntrl, or isascii Subroutines and localeconv Subroutine in AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 1.

strtoimax or strtoumax Subroutine

Purpose
Converts string to integer type.

Syntax
#include <inttypes.h>

intmax_t strtoimax (nptr, endptr, base);
const char *restrict nptr;
char **restrict endptr;
int base;

uintmax_t strtoumax (nptr, endptr, base);
const char *restrict nptr;
char **restrict endptr;
int base;

Description
The strtoimax and strtoumax subroutines are equivalent to the strtol, strtoll, strtoul, and strtoull subroutines, except that the initial portion of the string shall be converted to intmax_t and uintmax_t representation, respectively.

Parameters
nptr Points to the string to be converted.
endptr Points to the object where the final string is stored.
base Determines the value of the integer represented in some radix.
Return Values
The `strtoimax` and `stroumax` subroutines return the converted value, if any.

If no conversion could be performed, zero is returned.

If the correct value is outside the range of representable values, `{INTMAX_MAX}`, `{INTMAX_MIN}`, or `{UINTMAX_MAX}` is returned (according to the return type and sign of the value, if any), and the `errno` global variable is set to ERANGE.

Related Information
The "strtol, strtoul, strtoll, strtoull, or atoi Subroutine" on page 341.

`inttypes.h` in AIX 5L Version 5.3 Files Reference.

strtok_r Subroutine

Purpose
Breaks a string into a sequence of tokens.

Libraries
Thread-Safe C Library (libc_r.a)

Syntax
```
#include <string.h>
char *strtok_r (String, Separators, Pointer);
char *String;
const char *Separators;
char **Pointer;
```

Description

**Note:** The `strtok_r` subroutine is used in a multithreaded environment.

The `strtok_r` subroutine breaks the string pointed to by the `String` parameter into a sequence of tokens, each of which is delimited by a byte from the string pointed to by the `Separators` parameter. The `Pointer` parameter holds the information necessary for the `strtok_r` subroutine to perform scanning on the `String` parameter. In the first call to the `strtok_r` subroutine, the value passed as the `Pointer` parameter is ignored.

The first call in the sequence searches the `String` parameter for the first byte that is not contained in the current separator string pointed to by the `Separators` parameter. If no such byte is found, no tokens exist in the `String` parameter, and a null pointer is returned. If such a byte is found, it is the start of the first token. The `strtok_r` subroutine also updates the `Pointer` parameter with the starting address of the token following the first occurrence of the `Separators` parameter.

In subsequent calls, a null pointer should be passed as the first parameter to the `strtok_r` subroutine instead of the `String` parameter. Each subsequent call with a null pointer as the value of the first argument starts searching from the `Pointer` parameter, using it as the first token. Otherwise, the subroutine’s behavior does not change. The `strtok_r` subroutine would return successive tokens until no tokens remain. The `Separators` parameter may be different from one call to another.

Parameters

`String` Points to a string from which an operation returns results.
Error Codes
The **strtok**

### Related Information

The "strten, strchr, strrcsr, strpbrk, strspn, strcppn, strstr, **strtok**, or strsep Subroutine" on page 333.

Writing Reentrant and Thread-Safe Code in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

### strtol, strtoul, strtoll, strtoull, or atoi Subroutine

#### Purpose

Converts a string to a signed or unsigned long integer or long long integer.

#### Library

Standard C Library (libc.a)

#### Syntax

```c
#include <stdlib.h>

long strtol (const char *String, char **EndPointer, int Base);
const char *String;
char **EndPointer;
int Base;

unsigned long strtoul (const char *, char **EndPointer, int Base);
unsigned long long int strtoull (const char *, char **EndPointer, int Base);

int atoi (const char *String);
const char *String;
```

#### Description

The **strtol** subroutine returns a long integer whose value is represented by the character string to which the **String** parameter points. The **strtol** subroutine scans the string up to the first character that is inconsistent with the **Base** parameter. Leading white-space characters are ignored, and an optional sign may precede the digits.

The **strtoul** subroutine provides the same functions but returns an unsigned long integer.
The **strtol** and **strtoull** subroutines provide the same functions but return long long and unsigned long long integers, respectively.

The **atoi** subroutine is equivalent to the **strtol** subroutine where the value of the **EndPointer** parameter is a null pointer and the **Base** parameter is a value of 10.

If the value of the **EndPointer** parameter is not null, then a pointer to the character that ended the scan is stored in **EndPointer**. If an integer cannot be formed, the value of the **EndPointer** parameter is set to that of the **String** parameter.

If the **Base** parameter is a value between 2 and 36, the subject sequence’s expected form is a sequence of letters and digits representing an integer whose radix is specified by the **Base** parameter. This sequence is optionally preceded by a + (positive) or - (negative) sign. 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 the **Base** parameter are permitted. If the **Base** parameter has a value of 16, the characters 0x or 0X optionally precede the sequence of letters and digits, following the + (positive) or - (negative) sign if present.

If the value of the **Base** parameter is 0, the string determines the base. Thus, after an optional leading sign, a leading 0 indicates octal conversion, and a leading 0x or 0X indicates hexadecimal conversion. The default is to use decimal conversion.

### Parameters

- **String**: Points to the character string to be converted.
- **EndPointer**: Points to a character string that contains the first character not converted.
- **Base**: Specifies the base to use for the conversion.

### Return Values

Upon successful completion, the **strtol**, **strtoul**, **strtol**, and **strtoull** subroutines return the converted value. If no conversion could be performed, 0 is returned, and the **errno** global variable is set to indicate the error. If the correct value is outside the range of representable values, the **strtol** subroutine returns a value of **LONG_MAX** or **LONG_MIN** according to the sign of the value, while the **strtoul** subroutine returns a value of **ULONG_MAX**. The **strtol** subroutine returns a value of **LLONG_MAX** or **LLONG_MIN**, according to the sign of the value. The **strtoul** subroutine returns a value of **ULONG_MAX**, and the **strtoull** subroutine returns a value of **ULLONG_MAX**.

### Error Codes

The **strtol** and **strtoul** subroutines return the following error codes:

- **ERANGE**: The correct value of the converted number causes underflow or overflow.
- **EINVAL**: The value of the **Base** parameter is not valid.

### Related Information

The **atof**, **atolf**, **strtol**, or **strtoll** subroutine, **scanf**, **fscanf**, **sscanf**, or **wscanf**, **setlocale**, **wstrtol**, or **watol**, **wstrtol**, **watol**, or **watoi** subroutines return the following error codes:

- **Subroutines Overview** in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
strptime Subroutine

Purpose
Converts a character string to a time value.

Library
Standard C Library (libc.a)

Syntax
```c
#include <time.h>

char *strptime (Buf, Format, Tm)
const char *Buf, *Format;
struct tm *Tm;
```

Description
The `strptime` subroutine converts the characters in the `Buf` parameter to values that are stored in the `Tm` structure, using the format specified by the `Format` parameter.

Parameters
- **Buf**
  Contains the character string to be converted by the `strptime` subroutine.
- **Format**
  Contains format specifiers for the `strptime` subroutine. The `Format` parameter contains 0 or more specifiers. Each specifier is composed of one of the following elements:
  - One or more white-space characters
  - An ordinary character (neither % [percent sign] nor a white-space character)
  - A format specifier

Note: If more than one format specifier is present, they must be separated by white space or a non-percent/non-alphanumeric character. If the separator between format specifiers is other than white space, the `Buf` string should hold the same separator at the corresponding locations.

The `LC_TIME` category defines the locale values for the format specifiers. The following format specifiers are supported:

- **%a**
  Represents the weekday name, either abbreviated as specified by the `abday` statement or full as specified by the `day` statement.
- **%A**
  Represents the weekday name, either abbreviated as specified by the `abday` statement or full as specified by the `day` statement.
- **%b**
  Represents the month name, either abbreviated as specified by the `abmon` statement or full as specified by the `month` statement.
- **%B**
  Represents the month name, either abbreviated as specified by the `abmon` statement or full as specified by the `month` statement.
- **%c**
  Represents the date and time format defined by the `d_t_fmt` statement in the `LC_TIME` category.
- **%C**
  Represents the century number (0 through 99); leading zeros are permitted but not required.
- **%d**
  Represents the day of the month as a decimal number (01 to 31).
- **%D**
  Represents the date in `%m/%d/%y` format (for example, 01/31/91).
- **%e**
  Represents the day of the month as a decimal number (01 to 31).
- **%E**
  Represents the combined alternate era year and name, respectively, in `%o %N` format.
- **%h**
  Represents the month name, either abbreviated as specified by the `abmon` statement or full as specified by the `month` statement.
- **%H**
  Represents the 24-hour-clock hour as a decimal number (00 to 23).
%l Represents the 12-hour-clock hour as a decimal number (01 to 12).
%j Represents the day of the year as a decimal number (001 to 366).
%m Represents the month of the year as a decimal number (01 to 12).
%M Represents the minutes of the hour as a decimal number (00 to 59).
%n Represents any white space.
%N Represents the alternate era name.
%o Represents the alternate era year.
%p Represents the a.m. or p.m. string defined by the am_pm statement in the LC_TIME category.
%r Represents 12-hour-clock time with a.m./p.m. notation as defined by the t_fmt_ampm statement, usually in the format %I:%M:%S %p.
%S Represents the seconds of the minute as a decimal number (00 to 61). The decimal number range of 00 to 61 provides for leap seconds.
%t Represents any white space.
%T Represents 24-hour-clock time in the format %H:%M:%S (for example, 16:55:15).
%u Represents the week of the year as a decimal number (00 to 53). Sunday, or its equivalent as defined by the day statement, is considered the first day of the week for calculating the value of this field descriptor.
%w Represents the day of the week as a decimal number (0 to 6). Sunday, or its equivalent as defined by the day statement in the LC_TIME category, is considered to be 0 for calculating the value of this field descriptor.
%W Represents the week of the year as a decimal number (00 to 53). Monday, or its equivalent as defined by the day statement in the LC_TIME category, is considered the first day of the week for calculating the value of this field descriptor.
%x Represents the date format defined by the d_fmt statement in the LC_TIME category.
%x Represents the time format defined by the t_fmt statement in the LC_TIME category.
%y Represents the year within century.
%Y Represents the year as a decimal number (for example, 1989).
%Z Represents the time-zone name, if one can be determined (for example, EST). No characters are displayed if a time zone cannot be determined.
%% Specifies a % (percent sign) character.

Some format specifiers can be modified by the E and O modifier characters to indicate an alternative format or specification. If the alternative format or specification does not exist in the current locale, the behavior will be as if the unmodified format specifier were used. The following modified format specifiers are supported:

%Ec Represents the locale’s alternative appropriate date and time as defined by the era_d_t_fmt statement.
%EC Represents the base year (or other time period) in the locale’s alternative form as defined by the era statement under the era_name category of the current era.
%Ex Represents the alternative date as defined by the era_d_fmt statement.
%EX Represents the locale’s alternative time as defined by the era_t_fmt statement.
%Ey Represents the offset from the %EC format specifier (year only) in the locale’s alternative form.
%EY Represents the full alternative-year format.
%Od Represents the month using the locale’s alternative numeric symbols. Leading 0’s are permitted but not required.
%Oe Represents the month using the locale’s alternative numeric symbols. Leading 0’s are permitted but not required.
%OH Represents the hour in 24-hour-clock time using the locale’s alternative numeric symbols.
%O1 Represents the hour in 12-hour-clock time using the locale’s alternative numeric symbols.
%Om Represents the month using the locale’s alternative numeric symbols.
%OM Represents the minutes using the locale’s alternative numeric symbols.
%OS Represents the seconds using the locale’s alternative numeric symbols.
%OU Represents the week number of the year using the locale’s alternative numeric symbols. Sunday is considered the first day of the week. Use the rules corresponding to the %u format specifier.
%Ow Represents the day of the week using the locale's alternative numeric symbols. Sunday is considered the first day of the week.

%OW Represents the week number of the year using the locale's alternative numeric symbols. Monday is considered the first day of the week. Use the rules corresponding to the %W format specifier.

%Oy Represents the year (offset from %C) using the locale's alternative numeric symbols.

A format specification consisting of white-space characters is performed by reading input until the first nonwhite-space character (which is not read) or up to no more characters can be read.

A format specification consisting of an ordinary character is performed by reading the next character from the Buf parameter. If this character differs from the character comprising the directive, the directive fails and the differing character and any characters following it remain unread. Case is ignored when matching Buf items, such as month or weekday names.

A series of directives composed of %n format specifiers, %t format specifiers, white-space characters, or any combination of the three items is processed by reading up to the first character that is not white space (which remains unread), or until no more characters can be read.

Tm Specifies the structure to contain the output of the strftime subroutine. If a conversion fails, the contents of the Tm structure are undefined.

Return Values
If successful, the strftime subroutine returns a pointer to the character following the last character parsed. Otherwise, a null pointer is returned.

Related Information
The scanf (“scanf, fscanf, sscanf, or wsscanf Subroutine” on page 126) subroutine, “strfmon Subroutine” on page 328, strftime (“strftime Subroutine” on page 330) subroutine, time subroutine, wcsftime (“wcsftime Subroutine” on page 496) subroutine.

LC_TIME Category in the Locale Definition Source File Format in AIX 5L Version 5.3 Files Reference.

Subroutines, Example Programs, and Libraries in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.


stty or gtty Subroutine

Purpose
Sets or gets terminal state.

Library
Standard C Library (libc.a)

Syntax
#include <sgtty.h>
Description
These subroutines have been made obsolete by the `ioctl` subroutine.

The `stty` subroutine sets the state of the terminal associated with the `FileDescriptor` parameter. The `gtty` subroutine retrieves the state of the terminal associated with `FileDescriptor`. To set the state of a terminal, the calling process must have write permission.

Use of the `stty` subroutine is equivalent to the `ioctl` (``FileDescriptor``, TIOSETP, `Buffer`) subroutine, while use of the `gtty` subroutine is equivalent to the `ioctl` (``FileDescriptor``, TIOGETP, `Buffer`) subroutine.

Parameters

- `FileDescriptor` Specifies an open file descriptor.
- `Buffer` Specifies the buffer.

Return Values
If the `stty` or `gtty` subroutine is successful, a value of 0 is returned. Otherwise, a value of -1 is returned and the `errno` global variable is set to indicate the error.

Related Information
The `ioctl` subroutine.

The Input and Output Handling Programmer’s Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

swab Subroutine

Purpose
Copies bytes.

Library
Standard C Library (`libc.a`)

Syntax
```c
#include <unistd.h>

void swab (void *From, void *To, ssize_t NumberOfBytes);
```

346 Technical Reference, Volume 2: Base Operating System and Extensions
Description
The `swab` subroutine copies the number of bytes pointed to by the `NumberOfBytes` parameter from the location pointed to by the `From` parameter to the array pointed to by the `To` parameter, exchanging adjacent even and odd bytes.

The `NumberOfBytes` parameter should be even and nonnegative. If the `NumberOfBytes` parameter is odd and positive, the `swab` subroutine uses `NumberOfBytes - 1` instead. If the `NumberOfBytes` parameter is negative, the `swab` subroutine does nothing.

Parameters

- **From**: Points to the location of data to be copied.
- **To**: Points to the array to which the data is to be copied.
- **NumberOfBytes**: Specifies the number of even and nonnegative bytes to be copied.

Related Information

The `memcpy`, `memchr`, `memcmp`, `memmove`, or `memset` subroutine, `string` subroutine, `strlen, strchr, strchr,strupbrk, strspn, strsr, strtok, or strtol` subroutine.


Input and Output Handling Programmer's Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

swapoff Subroutine

Purpose
Deactivates paging or swapping to a designated block device.

Library
Standard C Library (`libc.a`)

Syntax
```
int swapoff (PathName)
```
```
char *PathName;
```

Description
The `swapoff` subroutine deactivates a block device or logical volume that is actively being used for paging and swapping. There must be sufficient space to satisfy the system's paging space requirements in the remaining devices after this device is deactivated or `swapoff` will fail. Sufficient space must accommodate the current system-wide paging space usage and the `npswarn` value. Refer to the `swap` command for information on current system-wide paging space usage. Refer to the `npswarn` tunable parameter of the `vmo` command, and Values for the npswarn and npsskill parameters for information on the `npswarn` value.

Parameters

- **PathName**: Specifies the full path name of the block device or logical volume.
Error Codes
If an error occurs, the `errno` global variable is set to indicate the error:

- **EBUSY**: The deactivation is already running.
- **EINTR**: The signal was received during the processing of a request.
- **ENODEV**: The `PathName` file does not exist.
- **ENOMEM**: No memory is available.
- **ENOSPC**: There is not enough space in other paging spaces to satisfy the system’s requirements.
- **ENOTBLK**: The device must be a block device or logical volume.
- **ENOTDIR**: A component of the `PathName` prefix is not a directory.
- **EPERM**: Caller does not have proper authority.

Other errors are from calls to the device driver’s **open** subroutine or **ioctl** subroutine.

Related Information
The `swapoff` command, `vmo` command.

Values for the `npwarn` and `npskill` parameters

The `Subroutines Overview` in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

swapon Subroutine

**Purpose**
Activates paging or swapping to a designated block device.

**Library**
Standard C Library (`libc.a`)

**Syntax**
```c
#include <sys/vminfo.h>

int swapon (char *PathName);
```

**Description**
The `swapon` subroutine makes the designated block device available to the system for allocation for paging and swapping.

The specified block device must be a logical volume on a disk device. The paging space size is determined from the current size of the logical volume.

**Parameters**
- `PathName` Specifies the full path name of the block device.
Error Codes
If an error occurs, the errno global variable is set to indicate the error:

EINTR Signal was received during processing of a request.
EINVAL Invalid argument (size of device is invalid).
ENOENT The PathName file does not exist.
ENOMEM The maximum number of paging space devices (16) are already defined, or no memory is available.
ENOTBLK Block device required.
ENOTDIR A component of the PathName prefix is not a directory.
ENXIO No such device address.

Other errors are from calls to the device driver’s open subroutine or ioctl subroutine.

Related Information
The swapoff subroutine swapqry subroutine.
The swapoff command, swapon command.
The Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

swapqry Subroutine

Purpose
Returns paging device status.

Library
Standard C Library (libc.a)

Syntax
#include <sys/vminfo.h>

int swapqry (PathName, Buffer)
    char *PathName;
    struct pginfo *Buffer;

Description
The swapqry subroutine returns information to a user-designated buffer about active paging and swap devices.

Parameters
PathName Specifies the full path name of the block device.
Buffer Points to the buffer into which the status is stored.

Return Values
The swapqry subroutine returns 0 if the PathName value is an active paging device. If the Buffer value is not null, it also returns status information.
Error Codes
If an error occurs, the subroutine returns -1 and the errno global variable is set to indicate the error, as follows:

- EFAULT: Buffer pointer is invalid.
- EINVAL: Invalid argument.
- EINTR: Signal was received while processing request.
- ENODEV: Device is not an active paging device.
- ENOENT: The PathName file does not exist.
- ENOTBLK: Block device required.
- ENOTDIR: A component of the PathName prefix is not a directory.
- ENXIO: No such device address.

Related Information
The swapoff subroutine, swapon subroutine.
The swapoff command, swapon command.

Subroutines Overview and Understanding Paging Space Programming Requirements in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

symlink Subroutine

Purpose
Makes a symbolic link to a file.

Library
Standard C Library (libc.a)

Syntax
#include <unistd.h>

int symlink (Path1, Path2);  
const char *Path1;  
const char *Path2;

Description
The symlink subroutine creates a symbolic link with the file named by the Path2 parameter, which refers to the file named by the Path1 parameter.

As with a hard link (described in the link subroutine), a symbolic link allows a file to have multiple 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 parameter need not exist when the link is created. In addition, a symbolic link can cross file system boundaries.

When a component of a path name refers to a symbolic link rather than a directory, the path name contained in the symbolic link is resolved. If the path name in the symbolic link starts with a / (slash), it is resolved relative to the root directory of the process. If the path name in the symbolic link does not start with / (slash), it is resolved relative to the directory that contains the symbolic link.
If the symbolic link is not the last component of the original path name, remaining components of the original path name are resolved from the symbolic-link point.

If the last component of the path name supplied to a subroutine refers to a symbolic link, the symbolic link path name may or may not be traversed. Most subroutines always traverse the link; for example, the chmod, chown, link, and open subroutines. The statx subroutine takes an argument that determines whether the link is to be traversed.

The following subroutines refer only to the symbolic link itself, rather than to the object to which the link refers:

- **mkdir**
  Fails with the EEXIST error code if the target is a symbolic link.

- **mknod**
  Fails with the EEXIST error code if a symbolic link exists with the same name as the target file as specified by the Path parameter in the mknod and mkfifo subroutines.

- **open**
  Fails with EEXIST error code when the O_CREAT and O_EXCL flags are specified and a symbolic link exists for the path name specified.

- **readlink** *(readlink Subroutine* on page 35)*
  Applies only to symbolic links.

- **rename** *(rename Subroutine* on page 55)*
  Renames the symbolic link if the file to be renamed (the FromPath parameter for the rename subroutine) is a symbolic link. If the new name (the ToPath parameter for the rename subroutine) refers to an existing symbolic link, the symbolic link is destroyed.

- **rmdir** *(rmdir Subroutine* on page 60)*
  Fails with the ENOTDIR error code if the target is a symbolic link.

- **symlink**
  Running this subroutine causes an error if a symbolic link named by the Path2 parameter already exists. A symbolic link can be created that refers to another symbolic link; that is, the Path1 parameter can refer to a symbolic link.

- **unlink** *(unlink Subroutine* on page 471)*
  Removes the symbolic link.

Since the mode of a symbolic link cannot be changed, its mode is ignored during the lookup process. Any files and directories referenced by a symbolic link are checked for access normally.

**Parameters**

- **Path1**
  Specifies the contents of the Path2 symbolic link. This value is a null-terminated string representing the object to which the symbolic link will point. Path1 cannot be the null value and cannot be more than PATH_MAX characters long. PATH_MAX is defined in the limits.h file.

- **Path2**
  Names the symbolic link to be created.

**Return Values**

Upon successful completion, the symlink subroutine returns a value of 0. If the symlink subroutine fails, a value of -1 is returned and the errno global variable is set to indicate the error.
Error Codes
The symlink subroutine fails if one or more of the following are true:

- **EEXIST**  
  Path2 already exists.

- **EACCES**  
  The requested operation requires writing in a directory with a mode that denies write permission.

- **EROFS**  
  The requested operation requires writing in a directory on a read-only file system.

- **ENOSPC**  
  The directory in which the entry for the symbolic link is being placed cannot be extended because there is no space left on the file system containing the directory.

- **EDQUOT**  
  The directory in which the entry for the new symbolic link is being placed cannot be extended or disk blocks could not be allocated for the symbolic link because the user’s or group’s quota of disk blocks on the file system containing the directory has been exhausted.

The symlink subroutine can be unsuccessful for other reasons. See "Base Operating System Error Codes For Services That Require Path-Name Resolution" for a list of additional errors.

Related Information
The `chown`, `fchown`, `chownx`, or `fchown` subroutine, `link` subroutine, `mkdir` subroutine, `mknod` subroutine, `open`, `open`, or `create` subroutine, `readlink` subroutine, `rename` subroutine, `rmdir` subroutine, `stat` subroutine, `statx`, `stat`, `lstat`, `fullstat`, `ffullstat`, `stat64`, `lstat64`, `fstat64`, or `lstat64x` subroutine, `unlink` subroutine.

The `ln` command.

The `limits.h` file.

Files, Directories, and File Systems for Programmers in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

**sync Subroutine**

**Purpose**
Updates all file systems.

**Library**
Standard C Library (`libc.a`)

**Syntax**

```c
#include <unistd.h>

void sync ( )
```

**Description**
The `sync` subroutine causes all information in memory that should be on disk to be written out. The writing, although scheduled, is not necessarily complete upon return from this subroutine. Types of information to be written include modified superblocks, i-nodes, data blocks, and indirect blocks.

The `sync` subroutine should be used by programs that examine a file system, such as the `df` and `fsck` commands.

If Network File System (NFS) is installed on your system, information in memory that relates to remote files is scheduled to be sent to the remote node.
syncvfs Subroutine

Purpose
Updates a filesystem.

Syntax

```c
#include <fscntl.h>

int syncvfs (char *vfsName, int command);
```

Description

The `syncvfs` subroutine behaves in 3 different manners depending on the granularity specified. In each case the `GFS_SYNCVFS` flag is checked and `VFS_SYNCVFS` or `VFS_SYNC` is called on the GFS and/or VFS specified. In each case the the `command` parameter is passed untouched. The cases are:

- If a NULL pointer is passed through the `vfsName` parameter, the `FS_SYNCVFS_ALL` level is assumed, and the call loops through each GFS in a similar manner to the sync call.
- If `FS_SYNCVFS_FSTYPE` is passed, the GFS is scanned and the names compared. The GFS with the correct name (if one exists) is called with its own GFS pointer and a null VFS pointer.
- If `FS_SYNCVFS_FS` is passed, the mount point is looked up and, if it exists, `VFS_SYNCVFS` is called with the GFS pointer and the VFS pointer of the filesystem found.

Parameters

`vfsName` Depending on the value of the `command` parameter, this can either be NULL, the name of a filesystem type (for example, "jfs", "j2") or the name of a filesystem, specified by mount point (for example, "/testj2").
Command is the mask of two options, a level and a granularity. The granularity can be one of:

- **FS_SYNCVFS_ALL**
  - sync every filesystem
- **FS_SYNCVFS_FSTYPE**
  - sync every filesystem of VFS type corresponding to `vfsName`
- **FS_SYNCVFS_FS**
  - sync specific filesystem at `vfsName`

The level can be one of:

- **FS_SYNCVFS.Try**
  - daemon heurstics
- **FS_SYNCVFS_FORCE**
  - user requested sync
- **FS_SYNCVFS_QUIESCE**
  - full filesystem quiesce

**Return Values**

Upon successful completion, the `syncvfs` subroutine returns 0. If unsuccessful, -1 is returned and the `errno` global variable is set.

**_sync_cache_range Subroutine**

**Purpose**

Synchronizes the I cache with the D cache.

**Library**

Standard C Library (`libc.a`)

**Syntax**

```c
void _sync_cache_range (eaddr, count)
  caddr_t eaddr;
  uint count;
```

**Description**

The `_sync_cache_range` subroutine synchronizes the I cache with the D cache, given an effective address and byte count. Programs performing instruction modification can call this routine to ensure that the most recent instructions are fetched for the address range.

**Parameters**

- `eaddr` Specifies the starting effective address of the address range.
- `count` Specifies the byte count of the address range.

**Related Information**

The **clflush** (Cache Line Flush) Instruction in *Assembler Language Reference.*
sysconf Subroutine

Purpose
Determines the current value of a specified system limit or option.

Library
Standard C Library (libc.a)

Syntax
#include <unistd.h>

long int sysconf (Name);

Description
The sysconf subroutine determines the current value of certain system parameters, the configurable system limits, or whether optional features are supported. The Name parameter represents the system variable to be queried.

Parameters
Name  Specifies which system variable setting should be returned. The valid values for the Name parameter are defined in the limits.h, time.h, and unistd.h files and are described below:

_SC_AIO_LISTIO_MAX  Maximum number of Input and Output operations that can be specified in a list Input and Output call.
_SC_AIO_MAX  Maximum number of outstanding asynchronous Input and Output operations.
_SCASYNCHRONOUS_IO  Implementation supports the Asynchronous Input and Output option.
_SC_ARG_MAX  Specifies the maximum byte length of the arguments for one of the exec functions, including environment data.
_SC_BC_BASE_MAX  Specifies the maximum number iibase and obase variables allowed by the bc command.
_SC_BC_DIM_MAX  Specifies the maximum number of elements permitted in an array by the bc command.
_SC_BC_SCALE_MAX  Specifies the maximum scale variable allowed by the bc command.
_SC_BC_STRING_MAX  Specifies the maximum length of a string constant allowed by the bc command.
_SC_CHILD_MAX  Specifies the number of simultaneous processes per real user ID.
_SC_CLK_TCK  Indicates the clock-tick increment as defined by the CLK_TCK in the time.h file.
_SC_COLL_WEIGHTS_MAX  Specifies the maximum number of weights that can be assigned to an entry of the LC_COLLATE keyword in the locale definition file.
_SC_DELAYTIMER_MAX  Maximum number of Timer expiration overruns.
_SC_EXPR_NEST_MAX  Specifies the maximum number of expressions that can be nested within parentheses by the expr command.
_SC_JOB_CONTROL  If this symbol is defined, job control is supported.
_SC_IOV_MAX  Specifies the maximum number of iovec structures one process has available for use with the readv and writev subroutines.
_SC_LARGE_PAGESIZE  Size (in bytes) of a large-page.
_SC_LINE_MAX  Specifies the maximum byte length of a command’s input line (either standard input or another file) when a command is described as processing text files. The length includes room for the trailing new-line character.
_SC_LOGIN_NAME_MAX  Maximum length of a login name.
_SC_MQ_OPEN_MAX    Maximum number of open message queue descriptors.
_SC_MQ_Prio_MAX    Maximum number of message priorities.
_SC_MEMLOCK        Implementation supports the Process Memory Locking option.
_SC_MEMLOCK_RANGE  Implementation supports the Range Memory Locking option.
_SC_MEMORY_PROTECTION Implementation supports the Memory Protection option.
_SC_MESSAGE_PASSING Implementation supports the Message Passing option.
_SC_NGROUPS_MAX    Specifies the maximum number of simultaneous supplementary group IDs per process.
_SC_OPEN_MAX       Specifies the maximum number of files that one process can have open at any one time.
_SC_PASS_MAX       Specifies the maximum number of significant characters in a password (not including the terminating null character).
_SC_PASS_MAX       Maximum number of significant bytes in a password.
_SC_PAGESIZE       Equivalent to _SC_PAGE_SIZE.
_SC_PAGE_SIZE      Size in bytes of a page.
_SC_PRIORITIZED_IO Implementation supports the Prioritized Input and Output option.
_SC_PRIORITY_SCHEDULING Implementation supports the Process Scheduling option.
_SC_RE_DUP_MAX     Specifies the maximum number of repeated occurrences of a regular expression permitted when using the \{ m, n \} interval notation.
_SC_RTSIG_MAX      Maximum number of Realtime Signals reserved for applications use.

_SC_REALTIME_SIGNALS  Implementation supports the Realtime Signals Extension option.
_SC_SAVED_IDS        If this symbol is defined, each process has a saved set-user ID and set-group ID.
_SC_SEM_NSEMS_MAX    Maximum number of Semaphores per process.
_SC_SEM_VALUE_MAX    Maximum value a Semaphore may have.
_SC_SEMAPHORES       Implementation supports the Semaphores option.
_SC_SHARED_MEMORY_OBJECTS Implementation supports the Shared Memory Objects option.
_SC_SIGQUEUE_MAX     Maximum number of signals a process may send and have pending at any time.
_SC_STREAM_MAX      Specifies the maximum number of streams that one process can have open simultaneously.
_SC_SYNCHRONIZED_IO Implementation supports the Synchronised Input and Output option.
_SC_TIMER_MAX       Maximum number of per-process Timers.
_SC_TIMERS          Implementation supports the Timers option.
_SC_TZNAME_MAX      Specifies the maximum number of bytes supported for the name of a time zone (not of the TZ value).
_SC_VERSION         Indicates that the version or revision number of the POSIX standard is implemented to indicate the 4-digit year and 2-digit month that the standard was approved by the IEEE Standards Board. This value is currently the long integer 198808.
_SC_XBS5_ILP32_OFF32 Implementation provides a C-language compilation environment with 32-bit int, long, pointer and off_t types.
_SC_XBS5_ILP32_OFFBIG Implementation provides a C-language compilation environment with 32-bit int, long and pointer types and an off_t type using at least 64 bits.
_SC_XBS5_LP64_OFF64  Implementation provides a C-language compilation environment with 32-bit int and 64-bit long, pointer and off_t types.
_SC_XBS5_LPBIG_OFFBIG Implementation provides a C-language compilation environment with an int type using at least 32 bits and long, pointer and off_t types using at least 64 bits.
_SC_XOPEN_CRYPT     Indicates that the system supports the X/Open Encryption Feature Group.
_SC_XOPEN_LEGACY    The implementation supports the Legacy Feature Group.
_SC_XOPEN_REALTIME  The implementation supports the X/Open Realtime Feature Group.
_SC_XOPEN_REALTIME_THREADS The implementation supports the X/Open Realtime Threads Feature Group.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SC_XOPEN_ENH_I18N</td>
<td>Indicates that the system supports the X/Open Enhanced Internationalization Feature Group.</td>
</tr>
<tr>
<td>_SC_XOPEN_SHM</td>
<td>Indicates that the system supports the X/Open Shared Memory Feature Group.</td>
</tr>
<tr>
<td>_SC_XOPEN_VERSION</td>
<td>Indicates that the version or revision number of the X/Open standard is implemented.</td>
</tr>
<tr>
<td>_SC_XOPEN_XCU_VERSION</td>
<td>Specifies the value describing the current version of the XCU specification.</td>
</tr>
<tr>
<td>_SC_ATEXIT_MAX</td>
<td>Specifies the maximum number of register functions for the atexit subroutine.</td>
</tr>
<tr>
<td>_SC_PAGE_SIZE</td>
<td>Specifies page-size granularity of memory.</td>
</tr>
<tr>
<td>_SC_AES_OS_VERSION</td>
<td>Indicates OSF AES version.</td>
</tr>
<tr>
<td>_SC_2_VERSION</td>
<td>Specifies the value describing the current version of POSIX.2.</td>
</tr>
<tr>
<td>_SC_2_C_BIND</td>
<td>Indicates that the system supports the C Language binding option.</td>
</tr>
<tr>
<td>_SC_2_C_CHAR_TERM</td>
<td>Indicates that the system supports at least one terminal type.</td>
</tr>
<tr>
<td>_SC_2_C_DEV</td>
<td>Indicates that the system supports the C Language Development Utilities Option.</td>
</tr>
<tr>
<td>_SC_2_C_VERSION</td>
<td>Specifies the value describing the current version of POSIX.2 with the C Language binding.</td>
</tr>
<tr>
<td>_SC_2_FORT_DEV</td>
<td>Indicates that the system supports the FORTRAN Development Utilities Option.</td>
</tr>
<tr>
<td>_SC_2_FORT_RUN</td>
<td>Indicates that the system supports the FORTRAN Development Utilities Option.</td>
</tr>
<tr>
<td>_SC_2_LOCALEDEF</td>
<td>Indicates that the system supports the creation of locales.</td>
</tr>
<tr>
<td>_SC_2_SW_DEV</td>
<td>Indicates that the system supports the Software Development Utilities Option.</td>
</tr>
<tr>
<td>_SC_2_UPE</td>
<td>Indicates that the system supports the User Portability Utilities Option.</td>
</tr>
<tr>
<td>_SC_NPROCESSORS_CONF</td>
<td>Number of processors configured.</td>
</tr>
<tr>
<td>_SC_NPROCESSORS_ONLN</td>
<td>Number of processors online.</td>
</tr>
<tr>
<td>_SC_THREAD_DATAKEYS_MAX</td>
<td>Maximum number of data keys that can be defined in a process.</td>
</tr>
<tr>
<td>_SC_THREAD_DESTRUCTOR_ITERATIONS</td>
<td>Maximum number attempts made to destroy a thread’s thread-specific data.</td>
</tr>
<tr>
<td>_SC_THREAD_KEYS_MAX</td>
<td>Maximum number of data keys per process.</td>
</tr>
<tr>
<td>_SC_THREAD_STACK_MIN</td>
<td>Minimum value for the threads stack size.</td>
</tr>
<tr>
<td>_SC_THREAD_THREADS_MAX</td>
<td>Maximum number of threads within a process.</td>
</tr>
<tr>
<td>_SC_REENTRANT_FUNCTIONS</td>
<td>System supports reentrant functions (reentrant functions must be used in multi-threaded applications).</td>
</tr>
<tr>
<td>_SC_THREADS</td>
<td>System supports POSIX threads.</td>
</tr>
<tr>
<td>_SC_THREAD_ATTR_STACKADDR</td>
<td>System supports the stack address option for POSIX threads (stackaddr attribute of threads).</td>
</tr>
<tr>
<td>_SC_THREAD_ATTR_STACKSIZE</td>
<td>System supports the stack size option for POSIX threads (stacksize attribute of threads).</td>
</tr>
<tr>
<td>_SC_THREAD_PRIORITY_SCHEDULING</td>
<td>System supports the priority scheduling for POSIX threads.</td>
</tr>
<tr>
<td>_SC_THREAD_PRIO_INHERIT</td>
<td>System supports the priority inheritance protocol for POSIX threads (priority inversion protocol for mutexes).</td>
</tr>
<tr>
<td>_SC_THREAD_PRIO_PROTECT</td>
<td>System supports the priority ceiling protocol for POSIX threads (priority inversion protocol for mutexes).</td>
</tr>
<tr>
<td>_SC_THREAD_PROCESS_SHARED</td>
<td>System supports the process sharing option for POSIX threads (pshared attribute of mutexes and conditions).</td>
</tr>
<tr>
<td>_SC_TTY_NAME_MAX</td>
<td>Maximum length of a terminal device name.</td>
</tr>
</tbody>
</table>

**Note:** The _SYNCHRONIZED_IO, _SC_FSYNC, and _SC_MAPPED_FILES commands apply to operating system version 4.3 and later releases.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SC_SYNCHRONIZED_IO</td>
<td>Implementation supports the Synchronized Input and Output option.</td>
</tr>
<tr>
<td>_SC_FSYNC</td>
<td>Implementation supports the File Synchronization option.</td>
</tr>
<tr>
<td>_SC_MAPPED_FILES</td>
<td>Implementation supports the Memory Mapped Files option.</td>
</tr>
<tr>
<td>_SC_LPAR_ENABLED</td>
<td>Indicates whether LPARs are enabled or not.</td>
</tr>
<tr>
<td>_SC_AIX_KERNEL_BITMODE</td>
<td>Determines if the kernel is 32-bit or 64-bit.</td>
</tr>
<tr>
<td>_SC_AIX_REALMEM</td>
<td>Determines the amount of real memory in kilobytes.</td>
</tr>
</tbody>
</table>
The values returned for the variables supported by the system do not change during the lifetime of the process making the call.

**Return Values**

If the `sysconf` subroutine is successful, the current value of the system variable is returned. The returned value cannot be more restrictive than the corresponding value described to the application by the `limits.h`, `time.h`, or `unistd.h` file at compile time. The returned value does not change during the lifetime of the calling process. If the `sysconf` subroutine is unsuccessful, a value of -1 is returned.

**Error Codes**

If the `Name` parameter is invalid, a value of -1 is returned and the `errno` global variable is set to indicate the error. If the `Name` parameter is valid but is a variable not supported by the system, a value of -1 is returned, and the `errno` global variable is set to a value of `EINVAL`. If the system variable `_SC_AIX_MP_CAPABLE` is accessed by a non-root user, a value of -1 is returned and the `errno` global variable indicates the error.

**File**

`/usr/include/limits.h`

Contains system-defined limits.

**Related Information**

The `confstr` subroutine, `pathconf` subroutine.

The `bc` command, `expr` command.

Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

**sysconfig Subroutine**

**Purpose**

Provides a service for controlling system/kernel configuration.

**Library**

Standard C Library (`libc.a`)

**Syntax**

```c
#include <sys/types.h>
#include <sys/sysconfig.h>

int sysconfig (Cmd, Parmp, Parmlen)
int Cmd;
void *Parmp;
int Parmlen;
```

Description

The **sysconfig** subroutine is used to customize the operating system. This subroutine provides a means of loading, unloading, and configuring kernel extensions. These kernel extensions can be additional kernel services, system calls, device drivers, or file systems. The **sysconfig** subroutine also provides the ability to read and set system run-time operating parameters.

Use of the **sysconfig** subroutine requires appropriate privilege.

The particular operation that the **sysconfig** subroutine provides is defined by the value of the **Cmd** parameter. The following operations are defined:

- **SYS_KLOAD** ([“SYS_KLOAD sysconfig Operation” on page 364](#))
  - Loads a kernel extension object file into kernel memory.

- **SYS_SINGLELOAD** ([“SYS_SINGLELOAD sysconfig Operation” on page 372](#))
  - Loads a kernel extension object file only if it is not already loaded.

- **SYS_QUERYLOAD** ([“SYS_QUERYLOAD sysconfig Operation” on page 369](#))
  - Determines if a specified kernel object file is loaded.

- **SYS_KULOAD** ([“SYS_KULOAD sysconfig Operation” on page 367](#))
  - Unloads a previously loaded kernel object file.

- **SYS_QDVSW** ([“SYS_QDVSW sysconfig Operation” on page 368](#))
  - Checks the status of a device switch entry in the device switch table.

- **SYS_CFGDD** ([“SYS_CFGDD sysconfig Operation” on page 360](#))
  - Calls the specified device driver configuration routine (module entry point).

- **SYS_CFGKMOD** ([“SYS_CFGKMOD sysconfig Operation” on page 361](#))
  - Calls the specified module at its module entry point for configuration purposes.

- **SYS_GETPARMS** ([“SYS_GETPARMS sysconfig Operation” on page 364](#))
  - Returns a structure containing the current values of run-time system parameters found in the **var** structure.

- **SYS_SETPARMS** ([“SYS_SETPARMS sysconfig Operation” on page 370](#))
  - Sets run-time system parameters from a caller-provided structure.

- **SYS_GETLPARINFO** ([“SYS_GETLPARINFO sysconfig Operation” on page 363](#))
  - Copies the system LPAR information into a user-allocated buffer.

In addition, the **SYS_64BIT** flag can be bitwise or’ed with the **Cmd** parameter (if the **Cmd** parameter is **SYS_KLOAD** or **SYS_SINGLELOAD**). For kernel extensions, this indicates that the kernel extension does not export 64-bit system calls, but that all 32-bit system calls also work for 64-bit applications. For device drivers, this indicates that the device driver can be used by 64-bit applications.

“Loader Symbol Binding Support” [on page 365](#) explains the symbol binding support provided when loading kernel object files.
Parameters

Cmd Specifies the function that the sysconfig subroutine is to perform.
Parmp Specifies a user-provided structure.
Parmlen Specifies the length of the user-provided structure indicated by the Parmp parameter.

Return Values

These sysconfig operations return a value of 0 upon successful completion of the subroutine. Otherwise, a value of -1 is returned and the errno global variable is set to indicate the error.

Any sysconfig operation requiring a structure from the caller fails if the structure is not entirely within memory addressable by the calling process. A return value of -1 is passed back and the errno global variable is set toEFAULT.

Related Information

The ddconfig device driver entry point.


SYS_CFGDD sysconfig Operation

Purpose

Calls a previously loaded device driver at its module entry point.

Description

The SYS_CFGDD sysconfig operation calls a previously loaded device driver at its module entry point. The device driver's module entry point, by convention, is its ddconfig entry point. The SYS_CFGDD operation is typically invoked by device configure or unconfigure methods to initialize or terminate a device driver, or to request device vital product data.

The sysconfig subroutine puts no restrictions on the command code passed to the device driver. This allows the device driver's ddconfig entry point to provide additional services, if desired.

The parmp parameter on the SYS_CFGDD operation points to a cfg_dd structure defined in the sys/sysconfig.h file. The parmlen parameter on the sysconfig system call should be set to the size of this structure.

If the kmid variable in the cfg_dd structure is 0, the desired device driver is assumed to be already installed in the device switch table. The major portion of the device number (passed in the devno field in the cfg_dd structure) is used as an index into the device switch table. The device switch table entry indexed by this devno field contains the device driver's ddconfig entry point to be called.

If the kmid variable is not 0, it contains the module ID to use in calling the device driver. A uio structure is used to pass the address and length of the device-dependent structure, specified by the cfg_dd.ddsptr and cfg_dd.ddslen fields, to the device driver being called.

The ddconfig device driver entry point provides information on how to define the ddconfig subroutine.

The device driver to be called is responsible for using the appropriate routines to copy the device-dependent structure (DDS) from user to kernel space.
Return Values
If the SYS_CFGDD operation successfully calls the specified device driver, the return code from the ddconfig subroutine determines the value returned by this subroutine. If the ddconfig routine’s return code is 0, then the value returned by the sysconfig subroutine is 0. Otherwise the value returned is a -1, and the errno global variable is set to the return code provided by the device driver ddconfig subroutine.

Error Codes
Errors detected by the SYS_CFGDD operation result in the following values for the errno global variable:

- **EACCESS**: The calling process does not have the required privilege.
- **EFAULT**: The calling process does not have sufficient authority to access the data area described by the parmp and parmlen parameters provided on the system call. This error is also returned if an I/O error occurred when accessing data in this area.
- **EINVAL**: Invalid module ID.
- **ENODEV**: Module ID specified by the cfg_dd.kmid field was 0, and an invalid or undefined devno value was specified.

Related Information
The [sysconfig subroutine](#) on page 358

The ddconfig device driver entry point.

The uio structure.

SYS_CFGKMOD sysconfig Operation

Purpose
Invokes a previously loaded kernel object file at its module entry point.

Description
The SYS_CFGKMOD sysconfig operation invokes a previously loaded kernel object file at its module entry point, typically for initialization or termination functions. The SYS_CFGDD operation performs a similar function for device drivers.

The parmp parameter on the sysconfig subroutine points to a cfg_kmod structure, which is defined in the sys/sysconfig.h file. The kmid field in this structure specifies the kernel module ID of the module to invoke. This value is returned when using the SYS_KLOAD operation to load the object file.

The cmd field in the cfg_kmod structure is a module-dependent parameter specifying the action that the routine at the module's entry point should perform. This is typically used for initialization and termination commands after loading and prior to unloading the object file.

The mdiptr field in the cfg_kmod structure points to a module-dependent structure whose size is specified by the mdilen field. This field is used to provide module-dependent information to the module to be called. If no such information is needed, the mdiptr field can be null.
If the mdipt field is not null, then the **SYS_CFGKMOD** operation builds a **uio** structure describing the address and length of the module-dependent information in the caller's address space. The mdipt and mdil en fields are used to fill in the fields of this **uio** structure. The module is then called at its module entry point with the **cmd** parameter and a pointer to the **uio** structure. If there is no module-dependent information to be provided, the **uiop** parameter passed to the module's entry point is set to null.

The module's entry point should be defined as follows:

```c
int module_entry(cmd, uiop)
int cmd;
struct uio *uiop;
```

The definition of the module-dependent information and its length is specific to the module being configured. The called module is responsible for using the appropriate routines to copy the module-dependent information from user to kernel space.

### Return Values

If the kernel module to be invoked is successfully called, its return code determines the value that is returned by the **SYS_CFGKMOD** operation. If the called module's return code is 0, then the value returned by the **sysconfig** subroutine is 0. Otherwise the value returned is -1 and the **errno** global variable is set to the called module's return code.

### Error Codes

Errors detected by the **SYS_CFGKMOD** operation result in the following values for the **errno** global variable:

- **EINVAL** Invalid module ID.
- **EACCESS** The calling process does not have the required privilege.
- **EFAULT** The calling process does not have sufficient authority to access the data area described by the **parmp** and **parmlen** parameters provided on the system call. This error is also returned if an I/O error occurred when accessing data in this area.

### File

**sys/sysconfig.h** Contains structure definitions.

### Related Information

The **sysconfig** (**sysconfig Subroutine** on page 358) subroutine.

The **SYS_CFGDD** (**SYS_CFGDD sysconfig Operation** on page 360) sysconfig operation, **SYS_KLOAD** (**SYS_KLOAD sysconfig Operation** on page 364) sysconfig operation, **SYS_SINGLELOAD** (**SYS_SINGLELOAD sysconfig Operation** on page 372) sysconfig operation.

The **uio** structure.

[Device Configuration Subsystem Programming Introduction](#) in **AIX 5L Version 5.3 Kernel Extensions and Device Support Programming Concepts**

[Programming in the Kernel Environment Overview](#) in **AIX 5L Version 5.3 Kernel Extensions and Device Support Programming Concepts**

[Understanding Kernel Extension Binding](#) in **AIX 5L Version 5.3 Kernel Extensions and Device Support Programming Concepts**
**SYS_GETLPAR_INFO sysconfig Operation**

**Purpose**
Copies the system LPAR information into a user-allocated buffer.

**Description**
The **SYS_GETLPAR_INFO** sysconfig operation copies the system LPAR information into a user-allocated buffer.

The *parmp* parameter on the **sysconfig** subroutine points to a structure of type **getlpar_info**. Within the **getlpar_info** structure, the *lpar_namelen* field must be set by the user to the maximum length of the character buffer pointed to by *lpar_name*. On return, the *lpar_namelen* field will have its value replaced by the acual length of the *lpar_name* field. However, only the minimum of the actual length or the length provided by the user will be copied into the buffer pointed to by *lpar_name*. The *lpar_namesz*, *lpar_num*, and *lpar_name* fields will contain valid data on returning from the call only if the system is running as an LPAR as indicated by the value of the *lpar_flags* field being equal to **LPAR_ENABLED**.

If a value of 0 is specified for the *lpar_namesz* field, the partition name will not be copied out.

If the system is not an LPAR (namely it is running as an SMP system), but it is LPAR-capable, the **LPAR_CAPABLE** flag will be set on return.

The **getlpar_info** structure is defined below:

- **lpar_flags** signed short
- **LPAR_ENABLED**: System is LPAR enabled.
- **LPAR_CAPABLE**: System is LPAR capable, but running in SMP mode.
- **lpar_namesz** unsigned short
- **Size of partition name.**
- **lpar_num** int
- **Partition Number.**
- **lpar_name** char *
- **Partition Name.**

**Note:** The *parmlen* parameter (which is the third parameter to the **sysconfig** system call) is ignored by the **SYS_GETLPAR_INFO** sysconfig operation.

**Error Codes**
The **SYS_GETLPAR_INFO** operation returns a value of -1 if an error occurs and the **errno** global variable is set to one of the following error codes:

- **EFAULT** The calling process does not have sufficient authority to access the data area described by the *parmp* and *parmlen* parameters provided on the subroutine or the *lpar_name* field in the **getlpar_info** structure. This error is also returned if an I/O error occurred when accessing data in any of these areas.
- **EINVAL** Invalid command parameter to the **sysconfig** subroutine.

**Files**
- **sys/sysconfig.h** Contains structure definitions and flags.

**Related Information**
The “**sysconfig Subroutine**” on page 358 in **AIX 5L Version 5.3 Kernel Extensions and Device Support Programming Concepts**.
SYS_GETPARMS sysconfig Operation

Purpose
Copies the system parameter structure into a user-specified buffer.

Description
The **SYS_GETPARMS** sysconfig operation copies the system parameter **var** structure into a user-allocated buffer. This structure may be used for informational purposes alone or prior to setting specific system parameters.

In order to set system parameters, the **required fields in the var structure** must be modified, and then the **SYS_SETPARMS** ("SYS_SETPARMS sysconfig Operation" on page 370) operation can be called to change the system run-time operating parameters to the desired state.

The **parmp** parameter on the **sysconfig** subroutine points to a buffer that is to contain all or part of the **var** structure defined in the **sys/var.h** file. The fields in the **var_hdr** part of the **var** structure are used for parameter update control.

The **parmlen** parameter on the system call should be set to the length of the **var** structure or to the number of bytes of the structure that is desired. The complete definition of the system parameters structure can be found in the **sys/var.h** file.

Return Values
The **SYS_GETPARMS** operation returns a value of -1 if an error occurs and the **errno** global variable is set to one of the following error codes.

Error Codes
- **EACCES**  The calling process does not have the required privilege.
- **EFAULT**  The calling process does not have sufficient authority to access the data area described by the **parmp** and **parmlen** parameters provided on the subroutine. This error is also returned if an I/O error occurred when accessing data in this area.

File

**sys/var.h**  Contains structure definitions.

Related Information
The **sysconfig** ("sysconfig Subroutine" on page 358) subroutine and **sys_parm** ("sys_parm Subroutine" on page 379) subroutine.

The **SYS_SETPARMS** ("SYS_SETPARMS sysconfig Operation" on page 370) sysconfig operation.


SYS_KLOAD sysconfig Operation

Purpose
 Loads a kernel extension into the kernel.
Description

The SYS_KLOAD sysconfig operation is used to load a kernel extension object file specified by a path name into the kernel. A kernel module ID for that instance of the module is returned. The SYS_KLOAD operation loads a new copy of the object file into the kernel even though one or more copies of the specified object file may have already been loaded into the kernel. The returned module ID can then be used for any of these three functions:

- Subsequent invocation of the module’s entry point (using the SYS_CFGKMOD operation)
- Invocation of a device driver’s ddconfig subroutine (using the SYS_CFGDD operation)
- Unloading the kernel module (using the SYS_KULOAD operation).

The parmp parameter on the sysconfig subroutine must point to a cfg_load structure, (defined in the sys/sysconfig.h file), with the path field specifying the path name for a valid kernel object file. The parmlen parameter should be set to the size of the cfg_load structure.

Note: A separate sysconfig operation, the SYS_SINGLELOAD operation, also loads kernel extensions. This operation, however, only loads the requested object file if not already loaded.

Loader Symbol Binding Support

The following information describes the symbol binding support provided when loading kernel object files.

Importing Symbols

Symbols imported from the kernel name space are resolved with symbols that exist in the kernel name space at the time of the load. (Symbols are imported from the kernel name space by specifying the #!/unix character string as the first field in an import list at link-edit time.)

Kernel modules can also import symbols from other kernel object files. These other kernel object files are loaded along with the specified object file if they are required to resolve the imported symbols.

Finding Directory Locations for Unqualified File Names:

If the module header contains an unqualified base file name for the symbol (that is, no / [slash] characters in the name), a libpath search string is used to find the location of the shared object file required to resolve imported symbols. This libpath search string can be taken from one of two places. If the libpath field in the cfg_load structure is not null, then it points to a character string specifying the libpath to be used. However, if the libpath field is null, then the libpath is taken from the module header of the object file specified by the path field in the same cfg_load structure.

The libpath specification found in object files loaded in order to resolve imported symbols is not used.

The kernel loader service does not support deferred symbol resolution. The load of the kernel object file is terminated with an error if any imported symbols cannot be resolved.

Exporting Symbols

Any symbols exported by the specified kernel object file are added to the kernel name space. This makes these symbols available to other subsequently loaded kernel object files. Any symbols specified with the SYSCALL keyword in the export list at link-edit time are added to the system call table at load time. These symbols are then available to application programs as a system call. Symbols can be added to the 32-bit and 64-bit system call tables separately by using the syscall32 and syscall64 keywords. Symbols can be added to both system call tables by using the syscall3264 keyword. A kernel extension that just exports 32-bit system calls can have all its system calls exported to 64-bit as well by passing the SYS_64BIT flag or’ed with the SYS_KLOAD command to sysconfig.
Kernel object files loaded on behalf of the specified kernel object file to resolve imported symbols do not have their exported symbols added to the kernel name space.

These object files are considered private since they do not export symbols to the global kernel name space. For these types of object files, a new copy of the object file is loaded on each SYS_KLOAD operation of a kernel extension that imports symbols from the private object file. In order for a kernel extension to add its exported symbols to the kernel name space, it must be explicitly loaded with the SYS_KLOAD operation before any other object files using the symbols are loaded. For kernel extensions of this type (those exporting symbols to the kernel name space), typically only one copy of the object file should ever be loaded.

Return Values
If the object file is loaded without error, the module ID is returned in the kmid variable within the cfg_load structure and the subroutine returns a value of 0.

Error Codes
On error, the subroutine returns a value of -1 and the errno global variable is set to one of the following values:

- **EACCESS** One of the following reasons applies:
  - The calling process does not have the required privilege.
  - An object module to be loaded is not an ordinary file.
  - The mode of the object module file denies read-only permission.

- **EFAULT** The calling process does not have sufficient authority to access the data area described by the parmp and parmlen parameters provided on the system call. This error is also returned if an I/O error occurred when accessing data in this area.

- **ENOEXEC** The program file has the appropriate access permission, but has an invalid XCOFF object file indication in its header. The SYS_KLOAD operation only supports loading of XCOFF object files. This error is also returned if the loader is unable to resolve an imported symbol.

- **EINVAL** The program file has a valid XCOFF indicator in its header, but the header is damaged or is incorrect for the machine on which the file is to be run.

- **ENOMEM** The load requires more kernel memory than is allowed by the system-imposed maximum.

- **ETXTBSY** The object file is currently open for writing by some process.

File
sys/sysconfig.h Contains structure definitions.

Related Information
The sysconfig subroutine.

The SYS_SINGLELOAD (“SYS_SINGLELOAD sysconfig Operation” on page 372) sysconfig operation, SYS_KLOAD (“SYS_KLOAD sysconfig Operation” on page 367) sysconfig operation, SYS_CFGDD (“SYS_CFGDD sysconfig Operation” on page 360) sysconfig operation, SYS_CFGKMOD (“SYS_CFGKMOD sysconfig Operation” on page 361) sysconfig operation.

The ddconfig device driver entry point.
SYS_KULOAD sysconfig Operation

Purpose
Unloads a loaded kernel object file and any imported kernel object files that were loaded with it.

Description
The SYS_KULOAD sysconfig operation unloads a previously loaded kernel file and any imported kernel object files that were automatically loaded with it. It does this by decrementing the load and use counts of the specified object file and any object file having symbols imported by the specified object file.

The parmp parameter on the sysconfig subroutine should point to a cfg_load structure, as described for the SYS_KLOAD ("SYS_KLOAD sysconfig Operation" on page 364) operation. The kmid field should specify the kernel module ID that was returned when the object file was loaded by the SYS_KLOAD or SYS_SINGLELOAD ("SYS_SINGLELOAD sysconfig Operation" on page 372) operation. The path and libpath fields are not used for this command and can be set to null. The parmlen parameter should be set to the size of the cfg_load structure.

Upon successful completion, the specified object file (and any other object files containing symbols that the specified object file imports) will have their load and use counts decremented. If there are no users of any of the module’s exports and its load count is 0, then the object file is immediately unloaded.

However, if there are users of this module (that is, modules bound to this module’s exported symbols), the specified module is not unloaded. Instead, it is unloaded on some subsequent unload request, when its use and load counts have gone to 0. The specified module is not in fact unloaded until all current users have been unloaded.

Notes:
1. Care must be taken to ensure that a subroutine has freed all of its system resources before being unloaded. For example, a device driver is typically prepared for unloading by using the SYS_CFGDD ("SYS_CFGDD sysconfig Operation" on page 360) operation and specifying termination.
2. If the use count is not 0, and you cannot force it to 0, the only way to terminate operation of the kernel extension is to reboot the machine.

"Loader Symbol Binding Support" on page 365 explains the symbol binding support provided when loading kernel object files.

Return Values
If the unload operation is successful or the specified object file load count is successfully decremented, a value of 0 is returned.

Error Codes
On error, the specified file and any imported files are not unloaded, nor are their load and use counts decremented. A value of -1 is returned and the errno global variable is set to one of the following:

- EACCESS: The calling process does not have the required privilege.
- EINVAL: Invalid module ID or the specified module is no longer loaded or already has a load count of 0.
- EFAULT: The calling process does not have sufficient authority to access the data area described by the parmp and parmlen parameters provided to the subroutine. This error is also returned if an I/O error occurred when accessing data in this area.
Related Information


The sysconfig (["sysconfig Subroutine" on page 358] subroutine.


SYS_QDVSW sysconfig Operation

Purpose
Checks the status of a device switch entry in the device switch table.

Description
The SYS_QDVSW sysconfig operation checks the status of a device switch entry in the device switch table.

The parmp parameter on the sysconfig subroutine points to a qry_devsw structure defined in the sys/sysconfig.h file. The parmlen parameter on the subroutine should be set to the length of the qry_devsw structure.

The qry_devsw field in the qry_devsw structure is modified to reflect the status of the device switch entry specified by the qry_devsw field. (Only the major portion of the devno field is relevant.) The following flags can be returned in the status field:

- **DSW_UNDEFINED**
  - The device switch entry is not defined if this flag has a value of 0 on return.
- **DSW_DEFINED**
  - The device switch entry is defined.
- **DSW_CREAD**
  - The device driver in this device switch entry provides a routine for character reads or raw input. This flag is set when the device driver provides a ddread entry point.
- **DSW_CWRITE**
  - The device driver in this device switch entry provides a routine for character writes or raw output. This flag is set when the device driver provides a ddwrite entry point.
- **DSW_BLOCK**
  - The device switch entry is defined by a block device driver. This flag is set when the device driver provides a ddstrategy entry point.
- **DSW_MPX**
  - The device switch entry is defined by a multiplexed device driver. This flag is set when the device driver provides a ddmpx entry point.
- **DSW_SELECT**
  - The device driver in this device switch entry provides a routine for handling the select (["select Subroutine" on page 139] or poll subroutines. This flag is set when the device driver provides a ddselect entry point.
- **DSW_DUMP**
  - The device driver defined by this device switch entry provides the capability to support one or more of its devices as targets for a kernel dump. This flag is set when the device driver has provided a dddump entry point.
- **DSW_CONSOLE**
  - The device switch entry is defined by the console device driver.
- **DSW_TCPATH**
  - The device driver in this device switch entry supports devices that are considered to be in the trusted computing path and provides support for the revoke (["revoke Subroutine" on page 58] and frevoke subroutines. This flag is set when the device driver provides a ddrevoke entry point.
DSW_OPENED The device switch entry is defined and the device has outstanding opens. This flag is set when the device driver has at least one outstanding open.

The DSW_UNDEFINED condition is indicated when the device switch entry has not been defined or has been defined and subsequently deleted. Multiple status flags may be set for other conditions of the device switch entry.

Return Values
If no error is detected, this operation returns with a value of 0. If an error is detected, the return value is set to a value of -1.

Error Codes
When an error is detected, the errno global variable is also set to one of the following values:

- EACCESS The calling process does not have the required privilege.
- EINVAL Device number exceeds the maximum allowed by the kernel.
- EFAULT The calling process does not have sufficient authority to access the data area described by the parmp and parmlen parameters provided on the system call. This error is also returned if an I/O error occurred when accessing data in this area.

File
sys/sysconfig.h Contains structure definitions.

Related Information
The sysconfig("sysconfig Subroutine" on page 358) subroutine.

The ddread, device driver entry point, ddwrite, device driver entry point, ddstrategy, device driver entry point, ddmpx, device driver entry point, ddselect, device driver entry point, dddump, device driver entry point.

The console special file.


SYS_QUERYLOAD sysconfig Operation

Purpose
Determines if a kernel object file has already been loaded.

Description
The SYS_QUERYLOAD sysconfig operation performs a query operation to determine if a given object file has been loaded. This object file is specified by the path field in the cfg_load structure passed in with the
parmp parameter. This operation utilizes the same cfg_load structure that is specified for the SYS_KLOAD ("SYS_KLOAD sysconfig Operation" on page 364) operation.

If the specified object file is not loaded, the kmid field in the cfg_load structure is set to a value of 0 on return. Otherwise, the kernel module ID of the module is returned in the kmid field. If multiple instances of the module have been loaded into the kernel, the module ID of the one most recently loaded is returned.

The libpath field in the cfg_load structure is not used for this option.

Note: A path-name comparison is done to determine if the specified object file has been loaded. However, this operation will erroneously return a not loaded condition if the path name to the object file is expressed differently than it was on a previous load request. "Loader Symbol Binding Support" on page 365 explains the symbol binding support provided when loading kernel object files.

Return Values
If the specified object file is found, the module ID is returned in the kmid variable within the cfg_load structure and the subroutine returns a 0. If the specified file is not found, a kmid variable of 0 is returned with a return code of 0.

Error Codes
On error, the subroutine returns a -1 and the errno global variable is set to one of the following values:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EACCES</td>
<td>The calling process does not have the required privilege.</td>
</tr>
<tr>
<td>EFAULT</td>
<td>The calling process does not have sufficient authority to access the data area described by the parmp and parmlen parameters provided on the subroutine. This error is also returned if an I/O error occurred when accessing data in this area.</td>
</tr>
<tr>
<td>EFAULT</td>
<td>The path parameter points to a location outside of the allocated address space of the process.</td>
</tr>
<tr>
<td>EIO</td>
<td>An I/O error occurred during the operation.</td>
</tr>
</tbody>
</table>

Related Information
The sysconfig ("sysconfig Subroutine" on page 358) subroutine.

The SYS_SINGLELOAD ("SYS_SINGLELOAD sysconfig Operation" on page 372) sysconfig operation, SYS_KLOAD ("SYS_KLOAD sysconfig Operation" on page 364) sysconfig operation.


SYS_SETPARMS sysconfig Operation

Purpose
Sets the kernel run-time tunable parameters.

Description
The SYS_SETPARMS sysconfig operation sets the current system parameters from a copy of the system parameter var structure provided by the caller. Only the run-time tunable parameters in the var structure can be set by this subroutine.
If the var_vers and var_gen values in the caller-provided structure do not match the var_vers and var_gen values in the current system var structure, no parameters are modified and an error is returned. The var_vers, var_gen, and var_size fields in the structure should not be altered. The var_vers value is assigned by the kernel and is used to insure that the correct version of the structure is being used. The var_gen value is a generation number having a new value for each read of the structure. This provides consistency between the data read by the SYS_GETPARMS (“SYS_GETPARMS sysconfig Operation” on page 364) operation and the data written by the SYS_SETPARMS operation.

The parmp parameter on the sysconfig subroutine points to a buffer that contains all or part of the var structure as defined in the sys/var.h file.

The parmlen parameter on the subroutine should be set either to the length of the var structure or to the size of the structure containing the parameters to be modified. The number of system parameters modified by this operation is determined either by the parmlen parameter value or by the var_size field in the caller-provided var structure. (The smaller of the two values is used.)

The structure provided by the caller must contain at least the header fields of the var structure. Otherwise, an error will be returned. Partial modification of a parameter in the var structure can occur if the caller's data area does not contain enough data to end on a field boundary. It is up to the caller to ensure that this does not happen.

Return Values
The SYS_SETPARMS sysconfig operation returns a value of -1 if an error occurred.

Error Codes
When an error occurs, the errno global variable is set to one of the following values:

EACCESS  The calling process does not have the required privilege.
EINVAL    One of the following error situations exists:
          • The var_vers version number of the provided structure does not match the version number of the current var structure.
          • The structure provided by the caller does not contain enough data to specify the header fields within the var structure.
          • One of the specified variable values is invalid or not allowed. On the return from the subroutine, the var_vers field in the caller-provided buffer contains the byte offset of the first variable in the structure that was detected in error.
EAGAIN    The var_gen generation number in the structure provided does not match the current generation number in the kernel. This occurs if consistency is lost between reads and writes of this structure. The caller should repeat the read, modify, and write operations on the structure.
EFAULT    The calling process does not have sufficient authority to access the data area described by the parmp and parmlen parameters provided to the subroutine. This error is also returned if an I/O error occurred when accessing data in this area.

File
sys/var.h     Contains structure definitions.

Related Information
The sysconfig “sysconfig Subroutine” on page 358 subroutine and sys_parm “sys_parm Subroutine” on page 379 subroutine.

The SYS_GETPARMS “SYS_GETPARMS sysconfig Operation” on page 364 sysconfig operation.
SYS_SINGLELOAD sysconfig Operation

Purpose
Loads a kernel extension module if it is not already loaded.

Description
The SYS_SINGLELOAD sysconfig operation is identical to the SYS_KLOAD operation, except that the SYS_SINGLELOAD operation loads the object file only if an object file with the same path name has not already been loaded into the kernel.

If an object file with the same path name has already been loaded, the module ID for that object file is returned in the kmid field and its load count incremented. If the object file is not loaded, this operation performs the load request exactly as defined for the SYS_KLOAD operation.

This option is useful in supporting global kernel routines where only one copy of the routine and its data can be present. Typically routines that export symbols to be added to the kernel name space are of this type.

Note: A path name comparison is done to determine if the same object file has already been loaded. However, this function will erroneously load a new copy of the object file into the kernel if the path name to the object file is expressed differently than it was on a previous load request.

“Loader Symbol Binding Support” explains the symbol binding support provided when loading kernel object files.

Return Values
The SYS_SINGLELOAD operation returns the same set of error codes that the SYS_KLOAD operation returns.

Related Information
The sysconfig subroutine.

The SYS_KLOAD sysconfig operation.

syslog, openlog, closelog, or setlogmask Subroutine

Purpose
Controls the system log.

Library
Standard C Library (libc.a)

Syntax
#include <syslog.h>
void openlog (const char *ID, int LogOption, Facility);

void syslog (int Priority, Value,...
const char *Value;

void closelog ( )

int setlogmask (int MaskPriority)

void bsdlog (int Priority, Value,...
const char *Value;

Description

Attention: Do not use the syslog, openlog, closelog, or setlogmask subroutine in a multithreaded environment. See the multithread alternatives in the syslog_r subroutine article. The syslog subroutine is not threadsafe; for threadsafe programs the syslog_r subroutine should be used instead.

The syslog subroutine writes messages onto the system log maintained by the syslogd command.

Note: Messages passed to syslog that are longer than 900 bytes may be truncated by syslogd before being logged.

The message is similar to the printf fmt string, with the difference that %m is replaced by the current error message obtained from the errno global variable. A trailing new-line can be added to the message if needed.

Messages are read by the syslogd command and written to the system console or log file, or forwarded to the syslogd command on the appropriate host.

If special processing is required, the openlog subroutine can be used to initialize the log file.

Messages are tagged with codes indicating the type of Priority for each. A Priority is encoded as a Facility, which describes the part of the system generating the message, and as a level, which indicates the severity of the message.

If the syslog subroutine cannot pass the message to the syslogd command, it writes the message on the /dev/console file, provided the LOG_CONS option is set.

The closelog subroutine closes the log file.

The setlogmask subroutine uses the bit mask in the MaskPriority parameter to set the new log priority mask and returns the previous mask.

The LOG_MASK and LOG_UPTO macros in the sys/syslog.h file are used to create the priority mask. Calls to the syslog subroutine with a priority mask that does not allow logging of that particular level of message causes the subroutine to return without logging the message.
Parameters

**ID**
Contains a string that is attached to the beginning of every message. The *Facility* parameter encodes a default facility from the previous list to be assigned to messages that do not have an explicit facility encoded.

**LogOption**
Specifies a bit field that indicates logging options. The values of *LogOption* are:

- **LOG_CONS**
  Sends messages to the console if unable to send them to the *syslogd* command. This option is useful in daemon processes that have no controlling terminal.

- **LOG_NDELAY**
  Opens the connection to the *syslogd* command immediately, instead of when the first message is logged. This option is useful for programs that need to manage the order in which file descriptors are allocated.

- **LOG_NOWAIT**
  Logs messages to the console without waiting for forked children. Use this option for processes that enable notification of child termination through *SIGCHLD*; otherwise, the *syslog* subroutine may block, waiting for a child process whose exit status has already been collected.

- **LOG_ODELAY**
  Delays opening until the *syslog* subroutine is called.

- **LOG_PID**
  Logs the process ID with each message. This option is useful for identifying daemons.

**Facility**
Specifies which of the following values generated the message:

- **LOG_AUTH**
  Indicates the security authorization system: the *login* command, the *su* command, and so on.

- **LOG_DAEMON**
  Logs system daemons.

- **LOG_KERN**
  Logs messages generated by the kernel. Kernel processes should use the *bsdlog* routine to generate *syslog* messages. The syntax of *bsdlog* is identical to *syslog*. The *bsdlog* messages can only be created by kernel processes and must be of *LOG_KERN* priority. The *syslog* subroutine cannot log *LOG_KERN* facility messages. Instead it will log *LOG_USER* facility messages.

- **LOG_LPR**
  Logs the line printer spooling system.

- **LOG_LOCAL0 through LOG_LOCAL7**
  Reserved for local use.

- **LOG_MAIL**
  Logs the mail system.

- **LOG_NEWS**
  Logs the news subsystem.

- **LOG_UUCP**
  Logs the UUCP subsystem.

- **LOG_USER**
  Logs messages generated by user processes. This is the default facility when none is specified.
Priority

Specifies the part of the system generating the message, and as a level, indicates the severity of the message. The level of severity is selected from the following list:

- **LOG_ALERT**
  Indicates a condition that should be corrected immediately; for example, a corrupted database.

- **LOG_CRIT**
  Indicates critical conditions; for example, hard device errors.

- **LOG_DEBUG**
  Displays messages containing information useful to debug a program.

- **LOG_EMERG**
  Indicates a panic condition reported to all users; system is unusable.

- **LOG_ERR**
  Indicates error conditions.

- **LOG_INFO**
  Indicates general information messages.

- **LOG_NOTICE**
  Indicates a condition requiring special handling, but not an error condition.

- **LOG_WARNING**
  Logs warning messages.

MaskPriority

Enables logging for the levels indicated by the bits in the mask that are set and disabled where the bits are not set. The default mask allows all priorities to be logged.

Value

Specifies the values given in the Value parameters and follows the same syntax as the `printf` subroutine `Format` parameter.

Examples

1. To log an error message concerning a possible security breach, such as the following, enter:
   
   ```
   syslog (LOG_ALERT, "who:internal error 23");
   ```

2. To initialize the log file, set the log priority mask, and log an error message, enter:
   
   ```
   openlog ("ftpd", LOG_PID, LOG_DAEMON);
   setlogmask (LOG_UPTO (LOG_ERR));
   syslog (LOG_INFO);
   ```

3. To log an error message from the system, enter:
   
   ```
   syslog (LOG_INFO | LOG_LOCAL2, "foobar error: %m");
   ```

Related Information

The `profile` subroutine.

The `prof` command.

The `syslogd` daemon.

`_end, _etext, or edata` identifiers.

Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

**syslog_r, openlog_r, closelog_r, or setlogmask_r Subroutine**

**Purpose**

Controls the system log.
Library
Standard C Library (libc.a)

Syntax
#include <syslog.h>

int syslog_r (Priority, SysLogData, Format, . . .)
int Priority;
struct syslog_data *SysLogData;
const char *Format;

int openlog_r (ID, LogOption, Facility, SysLogData)
const char *ID;
int LogOption;
int Facility;

struct syslog_data *SysLogData;
void closelog_r (SysLogData)
struct syslog_data *SysLogData;

int setlogmask_r (MaskPriority, SysLogData)
int MaskPriority;
struct syslog_data *SysLogData;

Description
The syslog_r subroutine writes messages onto the system log maintained by the syslogd daemon.

The messages are similar to the Format parameter in the printf subroutine, except that the %m field is replaced by the current error message obtained from the errno global variable. A trailing new-line character can be added to the message if needed.

Messages are read by the syslogd daemon and written to the system console or log file, or forwarded to the syslogd daemon on the appropriate host.

If a program requires special processing, you can use the openlog_r subroutine to initialize the log file.

The syslog_r subroutine takes as a second parameter a variable of the type struct syslog_data, which should be provided by the caller. When that variable is declared, it should be set to the SYSLOG_DATA_INIT value, which specifies an initialization macro defined in the sys/syslog.h file. Without initialization, the data structure used to support the thread safety is not set up and the syslog_r subroutine does not work properly.

Messages are tagged with codes indicating the type of Priority for each. A Priority is encoded as a Facility, which describes the part of the system generating the message, and as a level, which indicates the severity of the message.

If the syslog_r subroutine cannot pass the message to the syslogd daemon, it writes the message the /dev/console file, provided the LOG_CONS option is set.

The closelog_r subroutine closes the log file.

The setlogmask_r subroutine uses the bit mask in the MaskPriority parameter to set the new log priority mask and returns the previous mask.
The **LOG_MASK** and **LOG_UPTO** macros in the `sys/syslog.h` file are used to create the priority mask. Calls to the `syslog_r` subroutine with a priority mask that does not allow logging of that particular level of message causes the subroutine to return without logging the message.

Programs using this subroutine must link to the `libpthreads.a` library.

### Parameters

**Priority**

Specifies the part of the system generating the message and indicates the level of severity of the message. The level of severity is selected from the following list:

- A condition that should be corrected immediately, such as a corrupted database.
- A critical condition, such as hard device errors.
- A message containing information useful to debug a program.
- A panic condition reported to all users, such as an unusable system.
- An error condition.
- A general information message.
- A condition requiring special handling, other than an error condition.
- A warning message.

**SysLogData**

Specifies a structure that contains the following information:

- The file descriptor for the log file.
- The status bits for the log file.
- A string for tagging the log entry.
- The mask of priorities to be logged.
- The default facility code.
- The address of the local logger.

**Format**

Specifies the format, given in the same format as for the `printf` subroutine.

**ID**

Contains a string attached to the beginning of every message. The Facility parameter encodes a default facility from the previous list to be assigned to messages that do not have an explicit facility encoded.

**LogOption**

Specifies a bit field that indicates logging options. The values of `LogOption` are:

- **LOG_CONS**
  
  Sends messages to the console if unable to send them to the `syslogd` command. This option is useful in daemon processes that have no controlling terminal.

- **LOG_NDELAY**
  
  Opens the connection to the `syslogd` command immediately, instead of when the first message is logged. This option is useful for programs that need to manage the order in which file descriptors are allocated.

- **LOG_NOWAIT**
  
  Logs messages to the console without waiting for forked children. Use this option for processes that enable notification of child termination through `SIGCHLD`; otherwise, the `syslog` subroutine may block, waiting for a child process whose exit status has already been collected.

- **LOG_ODELAY**
  
  Delays opening until the `syslog` subroutine is called.

- **LOG_PID**
  
  Logs the process ID with each message. This option is useful for identifying daemons.
Facility

Specifies which of the following values generated the message:

**LOG_AUTH**
Indicates the security authorization system: the `login` command, the `su` command, and so on.

**LOG_DAEMON**
Logs system daemons.

**LOG_KERN**
Logs messages generated by the kernel. Kernel processes should use the `bsdlog` routine to generate `syslog` messages. The syntax of `bsdlog` is identical to `syslog`. The `bsdlog` messages can only be created by kernel processes and must be of LOG_KERN priority.

**LOG_LPR**
Logs the line printer spooling system.

**LOG_LOCAL0 through LOG_LOCAL7**
Reserved for local use.

**LOG_MAIL**
Logs the mail system.

**LOG_NEWS**
Logs the news subsystem.

**LOG_UUCP**
Logs the UUCP subsystem.

**LOG_USER**
Logs messages generated by user processes. This is the default facility when none is specified.
- Remote file systems, such as the Andrew File System (AFS®).
- The UUCP subsystem.
- Messages generated by user processes. This is the default facility when none is specified.

MaskPriority

Enables logging for the levels indicated by the bits in the mask that are set, and disables logging where the bits are not set. The default mask allows all priorities to be logged.

Return Values

0 Indicates that the subroutine was successful.
-1 Indicates that the subroutine was not successful. Moves an error code, indicating the specific error, into the `errno` global variable.

Error Codes

When the `syslog_r` subroutine is unsuccessful, the `errno` global variable can be set to the following values:

**EAGAIN**
Exceeds the limit on the total number of processes running either system-wide or by a single user, or the system does not have the resources necessary to create another process.

**EBADF**
The `syslogd` daemon is not active.

**ECONNRESET**
The `syslogd` daemon stopped during the operation.

**ENOBUFFS**
Buffer resources were not available.

**ENOMEM**
Not enough space exists for this process.

**ENOTCONN**
The `syslogd` daemon stopped during the operation.
If WLM is running, the limit on the number of processes or threads in the class might have been met.

Examples
1. To log an error message concerning a possible security breach, enter:
   \texttt{syslog\_r (LOG\_ALERT, syslog\_data\_struct, \"\%s\", \"who\:internal error 23\")};
2. To initialize the log file, set the log priority mask, and log an error message, enter:
   \texttt{openlog\_r ("ftpd", LOG\_PID, LOG\_DAEMON, syslog\_data\_struct)};
   \texttt{setlogmask\_r (LOG\_UPTO\(\text{LOG}\_ERR\), syslog\_data\_struct)};
   \texttt{syslog\_r (LOG\_INFO, syslog\_data\_struct, \"\")};
3. To log an error message from the system, enter:
   \texttt{syslog\_r (LOG\_INFO | LOG\_LOCAL2, syslog\_data\_struct, \"system error: %m\")};

Related Information
The \texttt{prof} command.
The \texttt{syslogd} daemon.
The \texttt{printf, fprintf, sprintf, wprintf, vprintf, vsprintf, or vwsprintf} subroutine.

Subroutines Overview and List of Multithread Subroutines in \textit{AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs}.

\section*{sys\_parm Subroutine}

\section*{Purpose}
Provides a service for examining or setting kernel run-time tunable parameters.

\section*{Library}
Standard C Library (\texttt{libc.a})

\section*{Syntax}
\begin{verbatim}
#include <sys/types.h>
#include <sys/var.h>

int sys_parm (cmd, parmflag, parmp)
int cmd;
int parmflag;
struct vario *parmp;
\end{verbatim}

\section*{Description}
The \texttt{sys\_parm} subroutine is used to query and/or customize run-time operating system parameters.

\textbf{Note:} This is a replacement service for \texttt{sysconf} with respect to querying or changing information in the \texttt{var} structure.

The \texttt{sys\_parm} subroutine:
\begin{itemize}
  \item Works on both 32 bit and 64 bit platforms
  \item Requires appropriate privilege for its use
\end{itemize}
The following operations are supported:

**SYSP_GET**
Returns a structure containing the current value of the specified run-time parameter found in the var structure.

**SYSP_SET**
Sets the value of the specified run-time parameter.

The run-time parameters that can be returned or set are found in the var structure as defined in var.h

### Parameters

- **cmd**
  Specifies the **SYSP_GET** or **SYSP_SET** function.

- **parmflag**
  Specifies the parameter upon which the function will act.

- **parmp**
  Points to the user specified structure from which or to which the system parameter value is copied. parmp points to a structure of type vario as defined in var.h.

The **vario** structure is an abstraction of the various fields in the var structure for which each field is size invariant. The size of the data does not depend on the execution environment of the kernel being 32 or 64 bit or the calling application being 32 or 64 bit.

### Examples

1. To examine the value of v.v_iostrun (collect disk usage statistics).

   ```c
   #include <sys/var.h>
   #include <stdio.h>
   struct vario myvar;
   rc=sys_parm(SYSP_GET,SYSP_V_IOSTRUN,);
   if(rc==0)
      printf("v.v_iostrun is set to %d\n",myvar.v.v_iostrun.value);
   ```

2. To change the value of v.v_iostrun (collect disk usage statistics).

   ```c
   #include <sys/var.h>
   #include <stdio.h>
   struct vario myvar;
   myvar.v.v_iostrun.value=0; /* initialize to false */
   rc=sys_parm(SYSP_SET,SYSP_V_IOSTRUN,);
   if(rc==0)
      printf("disk usage statistics are not being collected\n");
   ```

Other parameters may be examined or set by changing the parmflag parameter.

### Return Values

These operations return a value of 0 upon successful completion of the subroutine. Otherwise or a value of -1 is returned and the errno global variable is set to indicate the error.

### Error Codes

- **EACCES**
  The calling process does not have the required privilege.

- **EINVAL**
  One of the following is true:
  - The command is neither **SYSP_GET** nor **SYSP_SET**
  - parmflag is out of range of parameters defined in var.h
  - The value specified in the parmp parameter is not a valid value for the field indicated by the parmflag parameter.

- **EFAULT**
  An invalid address was specified by the parmp parameter.
**Related Information**

The **SYS_GETPARMS** (*SYS_GETPARMS sysconfig Operation* on page 364) **sysconfig** Operation, and the **SYS_SETPARMS** (*SYS_SETPARMS sysconfig Operation* on page 370) **sysconfig** Operation.

### System Subroutine

**Purpose**

Runs a shell command.

**Library**

Standard C Library (*libc.a*)

**Syntax**

```c
#include <stdlib.h>

int system (const char *String);
```

**Description**

The `system` subroutine passes the `String` parameter to the `sh` command as input. Then the `sh` command interprets the `String` parameter as a command and runs it.

The `system` subroutine calls the `fork` subroutine to create a child process that in turn uses the `exec` subroutine to run the `/usr/bin/sh` command, which interprets the shell command contained in the `String` parameter. When invoked on the Trusted Path, the `system` subroutine runs the Trusted Path shell (/usr/bin/tsh). The current process waits until the shell has completed, then returns the exit status of the shell. The exit status of the shell is returned in the same manner as a call to the `wait` or `waitpid` subroutine, using the structures in the `sys/wait.h` file.

The `system` subroutine ignores the `SIGINT` and `SIGQUIT` signals, and blocks the `SIGCHILD` signal while waiting for the command specified by the `String` parameter to terminate. If this might cause the application to miss a signal that would have killed it, the application should use the value returned by the `system` subroutine to take the appropriate action if the command terminated due to receipt of a signal. The `system` subroutine does not affect the termination status of any child of the calling process unless that process was created by the `system` subroutine. The `system` subroutine does not return until the child process has terminated.

**Parameters**

- `String` Specifies a valid `sh` shell command.

**Note:** The `system` subroutine runs only `sh` shell commands. The results are unpredictable if the `String` parameter is not a valid `sh` shell command.
Return Values
Upon successful completion, the system subroutine returns the exit status of the shell. The exit status of the shell is returned in the same manner as a call to the wait or waitpid subroutine, using the structures in the sys/wait.h file.

If the String parameter is a null pointer and a command processor is available, the system subroutine returns a nonzero value. If the fork subroutine fails or if the exit status of the shell cannot be obtained, the system subroutine returns a value of -1. If the exec I subroutine fails, the system subroutine returns a value of 127. In all cases, the errno global variable is set to indicate the error.

Error Codes
The system subroutine fails if any of the following are true:

EAGAIN The system-imposed limit on the total number of running processes, either systemwide or by a single user ID, was exceeded.
Eintr The system subroutine was interrupted by a signal that was caught before the requested process was started. The EINTR error code will never be returned after the requested process has begun.
ENOMEM Insufficient storage space is available.

Related Information
The exec subroutine, exit subroutine, fork subroutine, pipe subroutine, wait, “wait, waitpid, wait3, or waitpid Subroutine” on page 488 subroutine, wait3 subroutine, wait364 Subroutine” on page 488 subroutine.
The sh command.

List of Security and Auditing Subroutines, Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

tan, tanf, or tanl Subroutine

Purpose
Computes the tangent.

Syntax
```
#include <math.h>

float tanf (x)
float x;

long double tanl (x)
long double x;

double tan (x)
double x;
```

Description
The tan, tanf, and tanl subroutines compute the tangent of the x parameter, measured in radians.

An application wishing to check for error situations should set the errno global variable to zero and call feclearexcept(FE_ALL_EXCEPT) before calling these functions. Upon return, if errno is nonzero or fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is nonzero, an error has occurred.

382 Technical Reference, Volume 2: Base Operating System and Extensions
Parameters

\[ x \] Specifies the value to be computed.

Return Values

Upon successful completion, the `tan`, `tanhf`, and `tanhl` subroutines return the tangent of \( x \).

If the correct value would cause underflow, and is not representable, a range error may occur, and 0.0 is returned.

If \( x \) is NaN, a NaN is returned.

If \( x \) is ±0, \( x \) is returned.

If \( x \) is subnormal, a range error may occur and \( x \) should be returned.

If \( x \) is ±Inf, a domain error occurs, and a NaN returned.

If the correct value would cause underflow, and is representable, a range error may occur and the correct value is returned.

If the correct value would cause overflow, a range error occurs and the `tan`, `tanhf`, and `tanhl` subroutines return the value of the macro `HUGE_VAL`, `HUGE_VALF`, and `HUGE_VALL`, respectively.

Error Codes

The `tan`, `tanhf`, and `tanhl` subroutines lose accuracy when passed a large value for the \( x \) parameter. Since the machine value of \( \pi \) can only approximate its infinitely precise value, the remainder of \( x/(2 \times \pi) \) becomes less accurate as \( x \) becomes larger. Similar loss of accuracy occurs for the `tan`, `tanhf`, and `tanhl` subroutines during argument reduction of large arguments.

Related Information

`atanf` or `atanl` Subroutine, `fclearexcept` Subroutine, `fetestexcept` Subroutine, and `class`, `_class`, `finite`, `isnan`, or unordered Subroutines in AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 1.

`math.h` in AIX 5L Version 5.3 Files Reference.

```
tanh, tanhf, or tanhl Subroutine
```

Purpose

Computes the hyperbolic tangent.

Syntax

```c
#include <math.h>

float tanhf (float x);
long double tanhl (long double x);
```

Parameters

\[ x \] Specifies the value to be computed.
Description
The tanhf, tanhl, and tanh subroutines compute the hyperbolic tangent of the x.

An application wishing to check for error situations should set the errno global variable to zero and call feclerexcept(FE_ALL_EXCEPT) before calling these subroutines. Upon return, if errno is nonzero or fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is nonzero, an error has occurred.

Parameters
x Specifies the value to be computed.

Return Values
Upon successful completion, the tanhf, tanhl, and tanh subroutines return the hyperbolic tangent of x.

If x is NaN, a NaN is returned.
If x is ±0, x is returned.
If x is ±Inf, ±1 is returned.
If x is subnormal, a range error may occur and x should be returned.

Related Information
The "sin, sinf, or sinl Subroutine" on page 233.

Related Information
The "sin, sinf, or sinl Subroutine" on page 233.

atanh or atanhl Subroutine, feclearexcept Subroutine, fetestexcept Subroutine and class, _class, finite, isnan, or unordered Subroutines in AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 1.

math.h in AIX 5L Version 5.3 Files Reference.

tcb Subroutine

Purpose
Alters the Trusted Computing Base (TCB) status of a file.

Library
Security Library (libc.a)

Syntax
#include <sys/tcb.h>

int tcb (Path, Flag)
char *Path;
int Flag;

Description
The tcb subroutine provides a mechanism to query or set the TCB attributes of a file.
This subroutine is not safe for use with multiple threads. To call this subroutine from a threaded application, enclose the call with the _libs_rmutex lock. See “Making a Subroutine Safe for Multiple Threads” in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs for more information about this lock.

Parameters

Path Specifies the path name of the file whose TCB status is to be changed.
Flag Specifies the function to be performed. Valid values are defined in the sys/tcb.h file and include the following:

- **TCB_ON**: Enables the TCB attribute of a file.
- **TCB_OFF**: Disables the Trusted Process and TCB attributes of a file.
- **TCB_QUERY**: Queries the TCB status of a file. This function returns one of the preceding values.

Return Values

Upon successful completion, the tcb subroutine returns a value of 0 if the Flags parameter is either **TCB_ON** or **TCB_OFF**. If the Flags parameter is **TCB_QUERY**, the current status is returned. If the tcb subroutine fails, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes

The tcb subroutine fails if one of the following is true:

- **EINVAL**: The Flags parameter is not one of **TCB_ON**, **TCB_OFF**, or **TCB_QUERY**.
- **EPERM**: Not authorized to perform this operation.
- **ENOENT**: The file specified by the Path parameter does not exist.
- **EROFS**: The file system is read-only.
- **EBUSY**: The file specified by the Path parameter is currently open for writing.
- **EACCES**: Access permission is denied for the file specified by the Path parameter.

Security

Access Control: The calling process must have search permission for the object named by the Path parameter. Only the root user can set the tcb attributes of a file.

Related Information

The chmod or fchmod subroutine, statx, stat, lstat, fstatx, fstat, fullstat, or ffullstat subroutine.

The chmod command.

List of Security and Auditing Subroutines, Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

**tcdrain Subroutine**

**Purpose**

Waits for output to complete.
Library
Standard C Library (libc.a)

Syntax
#include <termios.h>

int tcdrain(FileDescriptor);

Description
The tcdrain subroutine waits until all output written to the object referred to by the FileDescriptor parameter has been transmitted.

Parameter
FileDescriptor Specifies an open file descriptor.

Return Values
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes
The tcdrain subroutine is unsuccessful if one of the following is true:

EBADF The FileDescriptor parameter does not specify a valid file descriptor.
EINTR A signal interrupted the tcdrain subroutine.
EIO The process group of the writing process is orphaned, and the writing process does not ignore or block the SIGTTOU signal.
ENOTTY The file associated with the FileDescriptor parameter is not a terminal.

Example
To wait until all output has been transmitted, enter:
rc = tcdrain(stdout);

Related Information
The tcflow subroutine, tcflush subroutine, tcsendbreak subroutine.

The Input and Output Handling Programmer’s Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

tcflow Subroutine

Purpose
Performs flow control functions.

Library
Standard C Library (libc.a)
## Syntax

```c
#include <termios.h>

int tcflow(int FileDescriptor, int Action);

int FileDescriptor;
int Action;
```

## Description

The `tcflow` subroutine suspends transmission or reception of data on the object referred to by the `FileDescriptor` parameter, depending on the value of the `Action` parameter.

## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileDescriptor</td>
<td>Specifies an open file descriptor.</td>
</tr>
<tr>
<td>Action</td>
<td>Specifies one of the following:</td>
</tr>
<tr>
<td></td>
<td>TCOOFF: Suspend output.</td>
</tr>
<tr>
<td></td>
<td>TCOON: Restart suspended output.</td>
</tr>
<tr>
<td></td>
<td>TCIOFF: Transmit a STOP character, which is intended to cause the terminal</td>
</tr>
<tr>
<td></td>
<td>device to stop transmitting data to the system. See the description of IXOFF</td>
</tr>
<tr>
<td></td>
<td>in the Input Modes section of the <code>termios.h</code> file.</td>
</tr>
<tr>
<td></td>
<td>TCION: Transmit a START character, which is intended to cause the terminal</td>
</tr>
<tr>
<td></td>
<td>device to start transmitting data to the system. See the description of IXOFF</td>
</tr>
<tr>
<td></td>
<td>in the Input Modes section of the <code>termios.h</code> file.</td>
</tr>
</tbody>
</table>

## Return Values

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the `errno` global variable is set to indicate the error.

## Error Codes

The `tcflow` subroutine is unsuccessful if one of the following is true:

- **EBADF**: The `FileDescriptor` parameter does not specify a valid file descriptor.
- **EINVAL**: The `Action` parameter does not specify a proper value.
- **EIO**: The process group of the writing process is orphaned, and the writing process does not ignore or block the `SIGTTOU` signal.
- **ENOTTY**: The file associated with the `FileDescriptor` parameter is not a terminal.

## Example

To restart output from a terminal device, enter:

```c
rc = tcflow(stdout, TCION);
```

## Related Information

The `tcdrain` subroutine, `tcflush` subroutine, `tcsendbreak` subroutine. The `Input and Output Handling Programmer's Overview` in `AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs`.
tcflush Subroutine

Purpose
Discards data from the specified queue.

Library
Standard C Library (libc.a)

Syntax
#include <termios.h>

int tcflush(int FileDescriptor, int QueueSelector);

Description
The tcflush subroutine discards any data written to the object referred to by the FileDescriptor parameter, or data received but not read by the object referred to by FileDescriptor, depending on the value of the QueueSelector parameter.

Parameters
- **FileDescriptor**: Specifies an open file descriptor.
- **QueueSelector**: Specifies one of the following:
  - TCIFLUSH: Flush data received but not read.
  - TCOFLUSH: Flush data written but not transmitted.
  - TCIOFLUSH: Flush both of the following:
    - Data received but not read
    - Data written but not transmitted

Return Values
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes
The tcflush subroutine is unsuccessful if one of the following is true:

- **EBADF**: The FileDescriptor parameter does not specify a valid file descriptor.
- **EINVAL**: The QueueSelector parameter does not specify a proper value.
- **EIO**: The process group of the writing process is orphaned, and the writing process does not ignore or block the SIGTTOU signal.
- **ENOTTY**: The file associated with the FileDescriptor parameter is not a terminal.
Example
To flush the output queue, enter:
\[ rc = tcflush(2, TCOFLUSH); \]

Related Information
The \texttt{tcdrain} ("tcdrain Subroutine" on page 385) subroutine, \texttt{tcflow} ("tcflow Subroutine" on page 386) subroutine, \texttt{tcsendbreak} ("tcsendbreak Subroutine" on page 391) subroutine.

The \textit{Input and Output Handling Programmer's Overview} in \textit{AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs}.

\textbf{tcgetattr Subroutine}

\textbf{Purpose}
Gets terminal state.

\textbf{Library}
Standard C Library (\texttt{libc.a})

\textbf{Syntax}
\begin{verbatim}
#include <termios.h>

int tcgetattr (FileDescriptor, TermiosPointer);

int FileDescriptor;
struct termios *TermiosPointer;
\end{verbatim}

\textbf{Description}
The \texttt{tcgetattr} subroutine gets the parameters associated with the object referred to by the \texttt{FileDescriptor} parameter and stores them in the \texttt{termios} structure referenced by the \texttt{TermiosPointer} parameter. This subroutine is allowed from a background process; however, the terminal attributes may subsequently be changed by a foreground process.

Whether or not the terminal device supports differing input and output baud rates, the baud rates stored in the \texttt{termios} structure returned by the \texttt{tcgetattr} subroutine reflect the actual baud rates, even if they are equal.

\textbf{Note}: If differing baud rates are not supported, returning a value of 0 as the input baud rate is obsolete.

\textbf{Parameters}
\begin{itemize}
  \item \texttt{FileDescriptor} \hspace{1cm} Specifies an open file descriptor.
  \item \texttt{TermiosPointer} \hspace{1cm} Points to a \texttt{termios} structure.
\end{itemize}

\textbf{Return Values}
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the \texttt{errno} global variable is set to indicate the error.
Error Codes
The tcgetattr subroutine is unsuccessful if one of the following is true:

EBADF      The FileDescriptor parameter does not specify a valid file descriptor.
ENOTTY     The file associated with the FileDescriptor parameter is not a terminal.

Examples
To get the current terminal state information, enter:
rc = tcgetattr(stdout, &my_termios);

Related Information
The tcsetattr subroutine.

The Input and Output Handling Programmer's Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

tcgetpgrp Subroutine

Purpose
Gets foreground process group ID.

Library
Standard C Library (libc.a)

Syntax
#include <unistd.h>

pid_t tcgetpgrp (int FileDescriptor);

Description
The tcgetpgrp subroutine returns the value of the process group ID of the foreground process group associated with the terminal. The function can be called from a background process; however, the foreground process can subsequently change the information.

Parameters

FileDescriptor      Indicates the open file descriptor for the terminal special file.

Return Values
Upon successful completion, the process group ID of the foreground process is returned. If there is no foreground process group, a value greater than 1 that does not match the process group ID of any existing process group is returned. Otherwise, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes
The tcgetpgrp subroutine is unsuccessful if one of the following is true:

EBADF      The FileDescriptor argument is not a valid file descriptor.
EINVAL  The function is not appropriate for the file associated with the `FileDescriptor` argument.
ENOTTY  The calling process does not have a controlling terminal or the file is not the controlling terminal.

Related Information
The `setpgid` subroutine, `setsid` subroutine, `tcsetpgrp` subroutine.
The `Input and Output Handling Programmer's Overview` in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

tcsendbreak Subroutine

Purpose
Sends a break on an asynchronous serial data line.

Library
Standard C Library (`libc.a`)

Syntax
```
#include <termios.h>

int tcsendbreak(int FileDescriptor, int Duration);
```

Description
If the terminal is using asynchronous serial data transmission, the `tcsendbreak` subroutine causes transmission of a continuous stream of zero-valued bits for a specific duration.

If the terminal is not using asynchronous serial data transmission, the `tcsendbreak` subroutine returns without taking any action.

Pseudo-terminals and LFT do not generate a break condition. They return without taking any action.

Parameters
- **FileDescriptor**: Specifies an open file descriptor.
- **Duration**: Specifies the number of milliseconds that zero-valued bits are transmitted. If the value of the `Duration` parameter is 0, it causes transmission of zero-valued bits for at least 250 milliseconds and not longer than 500 milliseconds. If `Duration` is not 0, it sends zero-valued bits for `Duration` milliseconds.

Return Values
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the `errno` global variable is set to indicate the error.

Error Codes
The `tcsendbreak` subroutine is unsuccessful if one or both of the following are true:
- **EBADF**: The `FileDescriptor` parameter does not specify a valid open file descriptor.
EIO  The process group of the writing process is orphaned, and the writing process does not ignore or block
the SIGTTOU signal.
ENOTTY  The file associated with the FileDescriptor parameter is not a terminal.

Examples
1. To send a break condition for 500 milliseconds, enter:
   `rc = tcsendbreak(stdout,500);`
2. To send a break condition for 25 milliseconds, enter:
   `rc = tcsendbreak(1,25);`
   This could also be performed using the default Duration by entering:
   `rc = tcsendbreak(1, 0);`

Related Information
The tcdrain subroutine, tcflow subroutine, tcflush subroutine.
The Input and Output Handling Programmer’s Overview in AIX 5L Version 5.3 General Programming
Concepts: Writing and Debugging Programs.

tcsetattr Subroutine

Purpose
Sets terminal state.

Library
Standard C Library (libc.a)

Syntax
```c
#include <termios.h>

int tcsetattr (FileDescriptor, OptionalActions, TermiosPointer)
int FileDescriptor, OptionalActions;
const struct termios * TermiosPointer;
```

Description
The tcsetattr subroutine sets the parameters associated with the object referred to by the FileDescriptor
parameter (unless support required from the underlying hardware is unavailable), from the termios
structure referenced by the TermiosPointer parameter.

The value of the OptionalActions parameter determines how the tcsetattr subroutine is handled.

The 0 baud rate (B0) is used to terminate the connection. If B0 is specified as the output baud rate when
the tcsetattr subroutine is called, the modem control lines are no longer asserted. Normally, this
disconnects the line.

Using 0 as the input baud rate in the termios structure to cause tcsetattr to change the input baud rate to
the same value as that specified by the value of the output baud rate, is obsolete.

If an attempt is made using the tcsetattr subroutine to set:
• An unsupported baud rate
• Baud rates, such that the input and output baud rates differ and the hardware does not support that combination
• Other features not supported by the hardware

but the `tcsetattr` subroutine is able to perform some of the requested actions, then the subroutine returns successfully, having set all supported attributes and leaving the above unsupported attributes unchanged.

If no part of the request can be honored, the `tcsetattr` subroutine returns a value of -1 and the `errno` global variable is set 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 the `tcgetattr` subroutine returns the actual state of the terminal device (reflecting both the changes made and not made in the previous `tcsetattr` call). The `tcsetattr` subroutine does not change the values in the `termios` structure whether or not it actually accepts them.

If the `tcsetattr` subroutine is called by a process which is a member of a background process group on a `FileDescriptor` associated with its controlling terminal, a `SIGTTOU` signal is sent to the background process group. If the calling process is blocking or ignoring `SIGTTOU` signals, the process performs the operation and no signal is sent.

**Parameters**

- **FileDescriptor**
  Specifies an open file descriptor.

- **OptionalActions**
  Specifies one of the following values:
  - **TCSANOW**
    The change occurs immediately.
  - **TCSADRAIN**
    The change occurs after all output written to the object referred to by `FileDescriptor` has been transmitted. This function should be used when changing parameters that affect output.
  - **TCSAFLUSH**
    The change occurs after all output written to the object referred to by `FileDescriptor` has been transmitted. All input that has been received but not read is discarded before the change is made.

- **TermiosPointer**
  Points to a `termios` structure.

**Return Values**

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the `errno` global variable is set to indicate the error.

**Error Codes**

The `tcsetattr` subroutine is unsuccessful if one of the following is true:

- **EBADF**
  The `FileDescriptor` parameter does not specify a valid file descriptor.
- **EINVAL**
  A signal interrupted the `tcsetattr` subroutine.
- **EINVAL**
  The `OptionalActions` argument is not a proper value, or an attempt was made to change an attribute represented in the `termios` structure to an unsupported value.
- **EIO**
  The process group of the writing process is orphaned, and the writing process does not ignore or block the `SIGTTOU` signal.
- **ENOTTY**
  The file associated with the `FileDescriptor` parameter is not a terminal.
Example
To set the terminal state after the current output completes, enter:
rc = tcsetattr(stdout, TCSADRAIN, &my_termios);

Related Information
The \texttt{cfgetispeed} subroutine, \texttt{tcgetattr} \texttt{"tcgetattr Subroutine" on page 389} subroutine.

The \texttt{Input and Output Handling Programmer's Overview} in \textit{AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs}.

\section*{tcsetpgrp Subroutine}

\subsection*{Purpose}
Sets foreground process group ID.

\subsection*{Library}
Standard C Library (\texttt{libc.a})

\subsection*{Syntax}
\begin{verbatim}
#include <unistd.h>

int tcsetpgrp (int FileDescriptor, pid_t ProcessGroupID);
\end{verbatim}

\subsection*{Description}
If the process has a controlling terminal, the \texttt{tcsetpgrp} subroutine sets the foreground process group ID associated with the terminal to the value of the \texttt{ProcessGroupID} parameter. The file associated with the \texttt{FileDescriptor} parameter 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 the \texttt{ProcessGroupID} parameter must match a process group ID of a process in the same session as the calling process.

\subsection*{Parameters}
\begin{tabular}{ll}
\texttt{FileDescriptor} & Specifies an open file descriptor. \\
\texttt{ProcessGroupID} & Specifies the process group identifier. \\
\end{tabular}

\subsection*{Return Values}
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the \texttt{errno} global variable is set to indicate the error.

\subsection*{Error Codes}
This function is unsuccessful if one of the following is true:
\begin{tabular}{ll}
\texttt{EBADF} & The \texttt{FileDescriptor} parameter is not a valid file descriptor. \\
\texttt{EINVAL} & The \texttt{ProcessGroupID} parameter is invalid. \\
\texttt{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. \\
\end{tabular}
The ProcessGroupID parameter is valid, but does not match the process group ID of a process in the same session as the calling process.

Related Information
The tcgetpgrp ("tcgetpgrp Subroutine" on page 390) subroutine.

The Input and Output Handling Programmer’s Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

termdf Subroutine

Purpose
Queries terminal characteristics.

Library
Standard C Library (libc.a)

Syntax
```c
char *termdf (FileDescriptor, Characteristic)
int FileDescriptor;
char Characteristic;
```

Description
The termdf subroutine returns a pointer to a null-terminated, static character string that contains the value of a characteristic defined for the terminal specified by the FileDescriptor parameter.

Asynchronous Terminal Support
Shell profiles usually set the TERM environment variable each time you log in. The stty command allows you to change the lines and columns (by using the lines and cols options). This is preferred over changing the LINES and COLUMNS environment variables, since the termdf subroutine examines the environment variables last. You consider setting LINES and COLUMNS environment variables if:

- You are using an asynchronous terminal and want to override the lines and cols setting in the terminfo database
- Your asynchronous terminal has an unusual number of lines or columns and you are running an application that uses the termdf subroutine but not an application which uses the terminfo database (for example, curses).

This is because the curses initialization subroutine, setupterm ("setupterm Subroutine" on page 674), calls the termdf subroutine to determine the number of lines and columns on the display. If the termdf subroutine cannot supply this information, the setupterm subroutine uses the values in the terminfo database.

Parameters

FileDescriptor Specifies an open file descriptor.
Characteristic

Specifies the characteristic that is to be queried. The following values can be specified:

\texttt{c} Causes the \texttt{termdef} subroutine to query for the number of “columns” for the terminal. This is determined by performing the following actions:
1. It requests a copy of the terminal’s \texttt{winsize} structure by issuing the \texttt{TIOCGWINSZ ioctl}. If \texttt{ws\_col} is not 0, the \texttt{ws\_col} value is used.
2. If the \texttt{TIOCGWINSZ ioctl} is unsuccessful or if \texttt{ws\_col} is 0, the \texttt{termdef} subroutine attempts to use the value of the \texttt{COLUMNS} environment variable.
3. If the \texttt{COLUMNS} environment variable is not set, the \texttt{termdef} subroutine returns a pointer to a null string.

\texttt{l} Causes the \texttt{termdef} subroutine to query for the number of “lines” (or rows) for the terminal. This is determined by performing the following actions:
1. It requests a copy of the terminal’s \texttt{winsize} structure by issuing the \texttt{TIOCGWINSZ ioctl}. If \texttt{ws\_row} is not 0, the \texttt{ws\_row} value is used.
2. If the \texttt{TIOCGWINSZ ioctl} is unsuccessful or if \texttt{ws\_row} is 0, the \texttt{termdef} subroutine attempts to use the value of the \texttt{LINES} environment variable.
3. If the \texttt{LINES} environment variable is not set, the \texttt{termdef} subroutine returns a pointer to a null string.

Characters other than \texttt{c} or \texttt{l} Cause the \texttt{termdef} subroutine to query for the “terminal type” of the terminal. This is determined by performing the following actions:
1. The \texttt{termdef} subroutine attempts to use the value of the \texttt{TERM} environment variable.
2. If the \texttt{TERM} environment variable is not set, the \texttt{termdef} subroutine returns a pointer to string set to “dumb”.

Examples

1. To display the terminal type of the standard input device, enter:
   \begin{verbatim}
   printf("\%s\n", termdef(0, 't'));
   \end{verbatim}
2. To display the current lines and columns of the standard output device, enter:
   \begin{verbatim}
   printf("lines\tcolumns\n\%s\t\%s\n", termdef(2, 'l'),
   termdef(2, 'c'));
   \end{verbatim}

   Note: If the \texttt{termdef} subroutine is unable to determine a value for lines or columns, it returns pointers to null strings.

Related Information

The \texttt{setupterm} ("setupterm Subroutine” on page 674) subroutine.

The \texttt{stty} command.

The Input and Output Handling Programmer’s Overview in \textit{AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs}.

\textbf{test\_and\_set Subroutine}

\textbf{Purpose}

Atomically tests and sets a memory location.
Library
Standard C library (libc.a)

Syntax
#include <sys/atomic_op.h>

boolean_t test_and_set(word_addr, mask)
atomic_p word_addr;
int mask;

Description
The test_and_set subroutine attempts to atomically OR the value stored at word_addr with the value specified by mask. If any bit in mask was already set in the value stored at word_addr, no update is made.

Parameters
word_addr Specifies the address of the memory location to be set.
mask Specifies the mask value to be used to set the memory location specified by word_addr.

Return Values
The test_and_set subroutine returns true if the value stored at word_addr was updated. Otherwise, it returns false.

Related Information
The fetch_and_and or fetch_and_or Subroutine in AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 1.

tgamma, tgammaf, or tgammal Subroutine

Purpose
Computes the gamma.

Syntax
#include <math.h>

double tgamma(x)
double x;

float tgammaf(x)
float x;

long double tgammal(x)
long double x;

Description
The tgamma, tgammaf, and tgammal subroutines compute the gamma function of x.

An application wishing to check for error situations should set errno to zero and call feclearexcept(FE_ALL_EXCEPT) before calling these subroutines. Upon return, if errno is nonzero or fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is nonzero, an error has occurred.
Parameters

$x$ Specifies the value to be computed.

Return Values

Upon successful completion, the $\text{tgamma}$, $\text{tgammaf}$, and $\text{tgammal}$ subroutines return $\text{Gamma}(x)$.

If $x$ is a negative integer, a domain error occurs, and either a NaN (if supported), or an implementation-defined value is returned.

If the correct value would cause overflow, a range error occurs and the $\text{tgamma}$, $\text{tgammaf}$, and $\text{tgammal}$ subroutines return the value of the macro $\text{HUGE_VAL}$, $\text{HUGE_VALF}$, or $\text{HUGE_VALL}$, respectively.

If $x$ is NaN, a NaN is returned.

If $x$ is +Inf, $x$ is returned.

If $x$ is ±0, a pole error occurs, and the $\text{tgamma}$, $\text{tgammaf}$, and $\text{tgammal}$ subroutines return ±$\text{HUGE_VAL}$, ±$\text{HUGE_VALF}$, and ±$\text{HUGE_VALL}$, respectively.

If $x$ is −Inf, a domain error occurs, and either a NaN (if supported), or an implementation-defined value is returned.

Related Information

$\text{feclearexcept}$ Subroutine, $\text{fetestexcept}$ Subroutine, and $\text{lgamma}$, $\text{lgammal}$, or $\text{gamma}$ Subroutine in AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 1.

$\text{math.h}$ in AIX 5L Version 5.3 Files Reference.

timer_create Subroutine

Purpose

Creates a per process timer.

Library

Standard C Library ($\text{libc.a}$)

Syntax

```c
#include <time.h>

int timer_create (clockid_t clock_id, sigevent *evp, timerid_t *timerid);
```

description

The $\text{timer_create}$ subroutine creates a per-process timer using the specified clock, $\text{clock_id}$, as the timing base. The $\text{timer_create}$ subroutine returns, in the location referenced by $\text{timerid}$, a timer ID of type $\text{timer_t}$ used to identify the timer in timer requests. This timer ID is unique within the calling process until the timer is deleted. The particular clock, $\text{clock_id}$, is defined in the $\text{time.h}$ file. The timer whose ID is returned is in a disarmed state upon return from the $\text{timer_create}$ subroutine.
The `evp` parameter, if non-NULL, points to a `sigevent` structure. This structure, allocated by the application, defines the asynchronous notification that will occur when the timer expires. If the `evp` parameter is NULL, the effect is as if the `evp` parameter pointed to a `sigevent` structure with the `sigev_notify` member having the value `SIGEV_SIGNAL`, the `sigev_signo` member having the `SIGALARM` default signal number, and the `sigev_value` member having the value of the timer ID.

This system defines a set of clocks that can be used as timing bases for per-process timers. Supported values for the `clock_id` parameter are the following:

- **CLOCK_REALTIME**: The system-wide realtime clock.
- **CLOCK_MONOTONIC**: The system-wide monotonic clock. The value of this clock represents the amount of time since an unspecified point in the past. It cannot be set through the `clock_settime` subroutine and cannot have backward clock jumps.
- **CLOCK_PROCESS_CPUTIME_ID**: The process CPU-time clock of the calling process. The value of this clock represents the amount of execution time of the process associated with the clock.
- **CLOCK_THREAD_CPUTIME_ID**: The thread CPU-time clock of the calling thread. The value of this clock represents the amount of execution time of the thread associated with this clock.

The `timer_create` subroutine fails if the value defined for the `clock_id` parameter corresponds to:

- The CPU-time clock of a process that is different than the process calling the function
- The thread CPU-time clock of a thread that is different than the thread calling the function.

### Parameters

- `clock_id`: Specifies the clock to be used.
- `evp`: Points to a `sigevent` structure that defines the asynchronous notification.
- `timerid`: Points to the location where the timer ID is returned.

### Return Values

If the `timer_create` subroutine succeeds, 0 is returned, and the location referenced by the `timerid` parameter is updated to a `timer_t`, which can be passed to the per-process timer calls. If an error occurs, -1 is returned and `errno` is set to indicate the error.

### Error Codes

The `timer_create` subroutine will fail if:

- **EAGAIN**: The system lacks sufficient signal queuing resources to honor the request.
- **EINVAL**: The specified clock ID is not defined.
- **ENOTSUP**: The implementation does not support the creation of a timer attached to the CPU-time clock that is specified by the `clock_id` parameter and associated with a process or a thread that is different from the process or thread calling `timer_create`.
- **ENOTSUP**: The function is not supported with checkpoint-restart processes.

### Related Information

- [“timer_delete Subroutine” on page 400](#)
- [“timer_getoverrun, timer_gettime, and timer_settime Subroutine” on page 400](#)
- [clock_getres](in AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 1).

---

Chapter 1. Base Operating System (BOS) Runtime Services (Q-Z) 399
**timer_delete Subroutine**

**Purpose**
Deletes a per process timer.

**Library**
Standard C Library (libc.a)

**Syntax**
```
#include <time.h>

int timer_delete (timerid)
timer_t timerid;
```

**Description**
The `timer_delete` subroutine deletes the specified timer, `timerid`, that was previously created by the `timer_create` subroutine. If the timer is armed when the `timer_delete` subroutine is called, the timer is automatically disarmed before removal.

**Parameters**
- `timerid` Specifies the timer ID.

**Return Values**
If successful, the `timer_delete` subroutine returns a value of zero. Otherwise, the subroutine returns a value of -1 and sets `errno` to indicate the error.

**Error Codes**
The `timer_delete` subroutine fails if:
- `EINVAL` The `timerid` parameter is not a valid timer ID.
- `ENOTSUP` The function is not supported with checkpoint-restart processes.

**Related Information**
"timer_create Subroutine" on page 398.

**timer_getovrrun, timer_gettime, and timer_settime Subroutine**

**Purpose**
Per-process timers.

**Library**
Standard C Library (libc.a)

**Syntax**
```
#include <time.h>

int timer_getovrrun (timerid)
timer_t timerid;
```

Technical Reference, Volume 2: Base Operating System and Extensions
Description

The `timer_gettime` subroutine stores the amount of time until the specified timer, `timerid`, expires, and stores the reload value of the timer into the space pointed to by the `value` parameter. The `it_value` member of the structure contains the amount of time before the timer expires, or zero if the timer is disarmed. This value is returned as the interval until the timer expires, even if the timer was armed with absolute time. The `it_interval` member of the `value` parameter contains the reload value last set by the `timer_settime` subroutine.

The `timer_settime` subroutine sets the time until the next expiration of the timer specified by the `timerid` parameter and arms the timer if the `it_value` member of the `value` parameter is nonzero. If the specified timer is armed when the `timer_settime` subroutine is called, the call resets the time until next expiration to the value specified. If the `it_value` member of the `value` parameter is zero, the timer is disarmed.

If the `TIMER_ABSTIME` flag is not set in the `flags` parameter, the `timer_settime` subroutine behaves as if the time until next expiration is set to be equal to the interval specified by the `it_value` member of the `value` parameter. That is, the timer expires in `it_value` nanoseconds from when the call is made. If the `TIMER_ABSTIME` flag is set in the `flags` parameter, the `timer_settime` subroutine behaves as if the time until next expiration is set to be equal to the difference between the absolute time specified by the `it_value` member and the current value of the clock associated with the `timerid` parameter. That is, the timer expires when the clock reaches the value specified by the `it_value` member. If the specified time has already passed, the subroutine succeeds and the expiration notification is made.

The reload value of the timer is set to the value specified by the `it_interval` member of the `value` parameter. When a timer is armed with a nonzero `it_interval`, a periodic (or repetitive) timer is specified.

Time values that are between two consecutive non-negative integer multiples of the resolution of the specified timer is rounded up to the larger multiple of the resolution. Quantization error does not cause the timer to expire earlier than the rounded time value.

If the `ovalue` parameter is not NULL, the `timer_settime` subroutine stores a value representing the previous amount of time before the timer would have expired, or zero if the timer was disarmed, together with the previous timer reload value. Timers do not expire before their scheduled time.

Only a single signal is queued to the process for a given timer at any point in time. When a timer for which a signal is still pending expires, no signal is queued, and a timer overrun occurs.

Concerning timers based on thread CPU-time clocks, the `timer_gettime` and `timer_settime` subroutines can only be called with `timerid` referencing a timer based on the thread CPU-time clock of the calling thread. In other words, a thread cannot manipulate the thread CPU-time timers created by other threads in the same process.

Parameters

- `timerid`: Specifies the timer ID.
- `value`: Points to an `itimerspec` structure containing the time value.
- `flags`: Specifies the flags that are set.

Description

The `timer_gettime` subroutine stores the amount of time until the specified timer, `timerid`, expires, and stores the reload value of the timer into the space pointed to by the `value` parameter. The `it_value` member of the structure contains the amount of time before the timer expires, or zero if the timer is disarmed. This value is returned as the interval until the timer expires, even if the timer was armed with absolute time. The `it_interval` member of the `value` parameter contains the reload value last set by the `timer_settime` subroutine.

The `timer_settime` subroutine sets the time until the next expiration of the timer specified by the `timerid` parameter and arms the timer if the `it_value` member of the `value` parameter is nonzero. If the specified timer is armed when the `timer_settime` subroutine is called, the call resets the time until next expiration to the value specified. If the `it_value` member of the `value` parameter is zero, the timer is disarmed.

If the `TIMER_ABSTIME` flag is not set in the `flags` parameter, the `timer_settime` subroutine behaves as if the time until next expiration is set to be equal to the interval specified by the `it_value` member of the `value` parameter. That is, the timer expires in `it_value` nanoseconds from when the call is made. If the `TIMER_ABSTIME` flag is set in the `flags` parameter, the `timer_settime` subroutine behaves as if the time until next expiration is set to be equal to the difference between the absolute time specified by the `it_value` member and the current value of the clock associated with the `timerid` parameter. That is, the timer expires when the clock reaches the value specified by the `it_value` member. If the specified time has already passed, the subroutine succeeds and the expiration notification is made.

The reload value of the timer is set to the value specified by the `it_interval` member of the `value` parameter. When a timer is armed with a nonzero `it_interval`, a periodic (or repetitive) timer is specified.

Time values that are between two consecutive non-negative integer multiples of the resolution of the specified timer is rounded up to the larger multiple of the resolution. Quantization error does not cause the timer to expire earlier than the rounded time value.

If the `ovalue` parameter is not NULL, the `timer_settime` subroutine stores a value representing the previous amount of time before the timer would have expired, or zero if the timer was disarmed, together with the previous timer reload value. Timers do not expire before their scheduled time.

Only a single signal is queued to the process for a given timer at any point in time. When a timer for which a signal is still pending expires, no signal is queued, and a timer overrun occurs.

Concerning timers based on thread CPU-time clocks, the `timer_gettime` and `timer_settime` subroutines can only be called with `timerid` referencing a timer based on the thread CPU-time clock of the calling thread. In other words, a thread cannot manipulate the thread CPU-time timers created by other threads in the same process.

Parameters

- `timerid`: Specifies the timer ID.
- `value`: Points to an `itimerspec` structure containing the time value.
- `flags`: Specifies the flags that are set.
ovalue Specifies the location of the value representing the previous amount of time before the timer would have expired, or zero if the timer was disarmed.

Return Values
If the timer_getoverrun subroutine succeeds, it returns the timer expiration overrun count.

If the timer_gettime or timer_settime subroutines succeed, 0 is returned.

If an error occurs for any of these subroutines, -1 is returned and errno is set to indicate the error.

Error Codes
The timer_getoverrun, timer_gettime, and timer_settime subroutines fail if:

EINVAL The timerid parameter does not correspond to an ID returned by the timer_create subroutine but not yet deleted by the timer_delete subroutine.
ENOTSUP The function is not supported with checkpoint-restart processes.

The timer_gettime and timer_settime subroutines fail if:

EINVAL The timerid parameter corresponds to a timer based on the thread CPU-time clock of a thread different from the thread calling timer_gettime or timer_settime. The timer has not been created by this thread.

The timer_settime subroutine fails if:

EINVAL The value parameter specified a nanosecond value less than zero or greater than or equal to 1000 million, and the it_value member of the structure did not specify zero seconds and nanoseconds.

Related Information
“timer_create Subroutine” on page 398.

clock_getres in AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 1

times Subroutine

Purpose
Gets process and waited-for child process times

Syntax
#include <sys/times.h>
clock_t times (buffer)
struct tms *buffer;

Description
The times subroutine fills the tms structure pointed to by buffer with time-accounting information. The tms structure 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 is included in the `tms_cutime` and `tms_cstime` elements of the parent when the `wait` or `waitpid` subroutine returns the process ID of the terminated child. If a child process has not waited for its children, their times are not 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.

Applications should use `sysconf(_SC_CLK_TCK)` to determine the number of clock ticks per second as it may vary from system to system.

**Parameters**

- `*buffer` Points to the `tms` structure.

**Return Values**

Upon successful completion, the `times` subroutine returns the elapsed real time, in clock ticks, since an arbitrary point in the past (for example, system startup time). This point does not change from one invocation of the `times` subroutine within the process to another. The return value may overflow the possible range of type `clock_t`. If the `times` subroutine fails, `(clock_t)-1` is returned, and the `errno` global variable is set to indicate the error.

**Examples**

**Timing a Database Lookup**

The following example defines two functions, `start_clock` and `end_clock`, that are used to time a lookup. It also defines variables of type `clock_t` and `tms` to measure the duration of transactions. The `start_clock` function saves the beginning times given by the `times` subroutine. The `end_clock` function gets the ending times and prints the difference between the two times.

```c
#include <sys/times.h>
#include <stdio.h>
...
void start_clock(void);
void end_clock(char *msg);
...
static clock_t st_time;
static clock_t en_time;
static struct tms st_cpu;
static struct tms en_cpu;
...
void start_clock()
{
    st_time = times(&st_cpu);
}

/* This example assumes that the result of each subtraction is within the range of values that can
   be represented in an integer type. */
void end_clock(char *msg)
{
    en_time = times(&en_cpu);
}
```
```c
fputs(msg,stdout);
printf("Real Time: %jd, User Time %jd, System Time %jd\n",
(intmax_t)(en_time - st_time),
(intmax_t)(en_cpu.tms_utime - st_cpu.tms_utime),
(intmax_t)(en_cpu.tms_stime - st_cpu.tms_stime));
}
```

Related Information

“sysconf Subroutine” on page 355 and “wait, waitpid, wait3, or wait364 Subroutine” on page 488

The `gettimer, settimer, restimer, stime, or time Subroutine`, `getinterval, incinterval, absinterval, resinc, resabs, alarm, ualarm, gettimer or settimer Subroutine`, `exec; execl, execle, execvp, execv, execve, execvp, or exec Subroutine` and `fork, f_fork, or vfork Subroutine` in AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 1.

---

timezone Subroutine

**Attention:** Do not use the `tzset` subroutine, from `libc.a`, when linking `libc.a libbsd.a`. The `tzset` subroutine uses the global external variable `timezone` which conflicts with the `timezone` subroutine in `libbsd.a`. This name collision can cause unpredictable results.

**Purpose**

Returns the name of the timezone associated with the first argument.

**Library**

Berkeley compatibility library (`libbsd.a`) (for `timezone` only)

**Syntax**

```c
#include <time.h>
char *timezone(zone, dst)
int zone;
int dst;
#include <time.h>
#include <limits.h>
int zone;
int dst;
char czone[TZNAME_MAX+1];
```

**Description**

The `timezone` subroutine returns the name of the timezone associated with the first argument which is measured in minutes westward from Greenwich. If the environment variable `TZ` is set, the first argument is ignored and the current timezone is calculated from the value of `TZ`. If the second argument is 0, the standard name is returned otherwise the Daylight Saving Time name is returned. If `TZ` is not set, then the internal table is searched for a matching timezone. If the timezone does not appear in the built in table then difference from GMT is produced.

Timezone returns a pointer to static data that will be overwritten by subsequent calls.

**Parameters**

- `zone` Specifies minutes westward from Greenwich.
- `dst` Specifies whether to return Standard time or Daylight Savings time.
- `czone` Specifies a buffer of size `TZNAME_MAX+1`, that the result is placed in.
Return Values

`timezone` returns a pointer to static data that contains the name of the timezone.

Errors

There are no errors defined.

Related Information

[Subroutines Overview](#)
[List of Multi-threaded Programming Subroutines](#)

---

### thread_post Subroutine

**Purpose**

Posts a thread of an event completion.

**Library**

Standard C library (`libc.a`)

**Syntax**

```c
#include <sys/thread.h>

int thread_post(tid_t tid);
```

**Description**

The `thread_post` subroutine posts the thread whose thread ID is indicated by the value of the `tid` parameter, of the occurrence of an event. If the posted thread is waiting in `thread_wait`, it will be awakened immediately. If it not waiting in `thread_wait`, the next call to `thread_wait` does not block but returns with success immediately.

Multiple posts to the same thread without an intervening wait by the specified thread will only count as a single post. The posting remains in effect until the indicated thread calls the `thread_wait` subroutine upon which the posting gets cleared.

The `thread_wait` and the `thread_post` subroutine can be used by applications to implement a fast IPC mechanism between threads in different processes.

**Parameters**

```
tid
```

Specifies the thread ID of the thread to be posted.

**Return Values**

On successful completion, the `thread_post` subroutine returns a value of 0. If unsuccessful, a value of -1 is returned and the global variable `errno` is set to indicate the error.

**Error Codes**

```
ESRCH
```

This indicated thread is non-existent or the thread has exited or is exiting.
The real or effective user ID does not match the real or effective user ID of the thread being posted, or else the calling process does not have root user authority.

Related Information
The thread_wait subroutine, and thread_post_many subroutine.

thread_post_many Subroutine

Purpose
Posts one or more threads of an event completion.

Library
Standard C library (libc.a)

Syntax
#include <sys/thread.h>

int thread_post_many(nthreads, tidp, erridp)

Parameters
nthreads
Specifies the number of threads to be posted.
tidp
Specifies the address of an array of thread IDs corresponding to the list of threads to be posted.
erridp
Either NULL or specifies the pointer to a thread ID variable in which the kernel will return the thread ID of the first failing thread when an errno of EPERM is set.

Return Values
On successful completion, the thread_post_many subroutine returns a value of 0. If unsuccessful, a value of -1 is returned and the global variable errno is set to indicate the error.
Error Codes

The `thread_post_many` subroutine is unsuccessful when one of the following is true:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESRCH</td>
<td>None of the indicated threads are existent or they have all exited or are exiting.</td>
</tr>
<tr>
<td>EPERM</td>
<td>The real or effective user ID does not match the real or effective user ID of one or more threads being posted, or else the calling process does not have root user authority.</td>
</tr>
<tr>
<td>EFAULT</td>
<td>The <code>tidp</code> parameter points to a location outside of the address space of the process.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>A negative value or a value greater than 512 was was specified in the <code>nthreads</code> parameter.</td>
</tr>
</tbody>
</table>

Related Information

The `thread_wait` subroutine, and `thread_post` subroutine.

### thread_self Subroutine

**Purpose**

Returns the caller’s kernel thread ID.

**Library**

Standard C library (`libc.a`)

**Syntax**

```c
#include <sys/thread.h>
tid_t thread_self()
```

**Description**

The `thread_self` subroutine returns the caller’s kernel thread ID. The kernel thread ID may be useful for the `bindprocessor` and `ptrace` subroutines. The `ps`, `trace`, and `vmstat` commands also report kernel thread IDs, thus this subroutine can be useful for debugging multi-threaded programs.

The kernel thread ID is unrelated with the thread ID used in the threads library (`libpthreads.a`) and returned by the `pthread_self` subroutine.

**Return Values**

The `thread_self` subroutine returns the caller’s kernel thread ID.

**Related Information**

The `bindprocessor` subroutine, `pthread_self` subroutine, `ptrace` subroutine.

### thread_setsched Subroutine

**Purpose**

Changes the scheduling policy and priority of a kernel thread.
Library
Standard C library (libc.a)

Syntax
#include <sys/sched.h>
#include <sys/pri.h>
#include <sys/types.h>

int thread_setsched (tid, priority, policy)
tid_t tid;
int priority;
int policy;

Description
The thread_setsched subroutine changes the scheduling policy and priority of a kernel thread. User threads (pthreads) have their own scheduling attributes that in some cases allow a pthread to execute on top of multiple kernel threads. Therefore, if the policy or priority change is being granted on behalf of a pthread, then the pthreads contention scope should be PTHREAD_SCOPE_SYSTEM.

Note: Caution must be exercised when using the thread_setsched subroutine, since improper use may result in system hangs. See sys/pri.h for restrictions on thread priorities.

Parameters
tid Specifies the kernel thread ID of the thread whose priority and policy are to be changed.
priority Specifies the priority to use for this kernel thread. The priority parameter is ignored if the policy is being set to SCHED_OTHER. The priority parameter must have a value in the range 0 to PRI_LOW. PRI_LOW is defined in sys/pri.h. See sys/pri.h for more information on thread priorities.
policy Specifies the policy to use for this kernel thread. The policy parameter can be one of the following values, which are defined in sys/sched.h:

SCHED_OTHER
Default operating system scheduling policy.

SCHED_FIFO
First in-first out scheduling policy.

SCHED_FIFO2
Allows a thread that sleeps for a relatively short amount of time to be requeued to the head, rather than the tail, of its priority run queue.

SCHED_FIFO3
Causes threads to be enqueued to the head of their run queues.

SCHED_RR
Round-robin scheduling policy.

Return Values
Upon successful completion, the thread_setsched subroutine returns a value of zero. If the thread_setsched subroutine is unsuccessful, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes
The thread_setsched subroutine is unsuccessful if one or more of the following is true:

ESRCH The kernel thread id tid is invalid.
EINVAL  The policy or priority is invalid.
EPERM  The caller does not have enough privilege to change the policy or priority.

thread_wait Subroutine

Purpose
Suspends the thread until it receives a post or times out.

Library
Standard C library (libc.a)

Syntax
#include <sys/thread.h>

int thread_wait(int timeout);

Description
The thread_wait subroutine allows a thread to wait or block until another thread posts it with the thread_post or the thread_post_many subroutine or until the time limit specified by the timeout value expires. It returns immediately if there is a pending post for this thread or if a timeout value of 0 is specified.

If the event for which the thread is waiting and for which it will be posted will occur only in the future, the thread_wait subroutine may be called with a timeout value of 0 to clear any pending posts.

The thread_wait and the thread_post subroutine can be used by applications to implement a fast IPC mechanism between threads in different processes.

Parameters

timeout Specifies the maximum length of time, in milliseconds, to wait for a posting. If the timeout parameter value is -1, the thread_wait subroutine does not return until a posting actually occurs. If the value of the timeout parameter is 0, the thread_wait subroutine does not wait for a post to occur but returns immediately, even if there are no pending posts. For a non-privileged user, the minimum timeout value is 10 msec and any value less than that is automatically increased to 10 msec.

Return Values
On successful completion, the thread_wait subroutine returns a value of 0. The thread_wait subroutine completes successfully if there was a pending post or if the calling thread was posted before the time limit specified by the timeout parameter expires.

A return value of THREAD_WAIT_TIMEDOUT indicates that the thread_wait subroutine timed out.

If unsuccessful, a value of -1 is returned and the global variable errno is set to indicate the error.
Error Codes
The thread_wait subroutine is unsuccessful when one of the following is true:

EINTR      This subroutine was terminated by receipt of a signal.
ENOMEM     There is not enough memory to allocate a timer.

Related Information
The thread_post subroutine and thread_post_many subroutine.

Tmpfile Subroutine

Purpose
Creates a temporary file.

Library
Standard C Library (libc.a)

Syntax
#include <stdio.h>
FILE *tmpfile()

Description
The tmpfile subroutine creates a temporary file and opens a corresponding stream. The file is opened for update. The temporary file is automatically deleted when all references (links) to the file have been closed.

The stream refers to a file which has been unlinked. If the process ends in the period between file creation and unlinking, a permanent file may remain.

Return Values
The tmpfile subroutine returns a pointer to the stream of the file that is created if the call is successful. Otherwise, it returns a null pointer and sets the errno global variable to indicate the error.

Error Codes
The tmpfile subroutine fails if one of the following occurs:

EINTR      A signal was caught during the tmpfile subroutine.
EMFILE     The number of file descriptors currently open in the calling process is already equal to OPEN_MAX.
ENFILE     The maximum allowable number of files is currently open in the system.
ENOSPEC    The directory or file system which would contain the new file cannot be expanded.

Related Information
The fopen, freopen, fdopen subroutines, mktemp subroutine, tmpnam or tempnam subroutine, unlink subroutine.
**tmpnam or tempnam Subroutine**

**Purpose**
Constructs the name for a temporary file.

**Library**
Standard C Library (libc.a)

**Syntax**
```c
#include <stdio.h>
char *tmpnam (String)
char *String;

char *tempnam (Directory, FileXPointer)
const char *Directory, *FileXPointer;
```

**Description**

**Attention:** The `tmpnam` and `tempnam` subroutines generate a different file name each time they are called. If called more than 16,384 (`TMP_MAX`) times by a single process, these subroutines recycle previously used names.

The `tmpnam` and the `tempnam` subroutines generate file names for temporary files. The `tmpnam` subroutine generates a file name using the path name defined as `P_tmpdir` in the `stdio.h` file.

Files created using the `tmpnam` subroutine reside in a directory intended for temporary use. The file names are unique. The application must create and remove the file.

The `tempnam` subroutine enables you to define the directory. The `Directory` parameter points to the name of the directory in which the file is to be created. If the `Directory` parameter is a null pointer or points to a string that is not a name for a directory, the path prefix defined as `P_tmpdir` in the `stdio.h` file is used. For an application that has temporary files with initial letter sequences, use the `FileXPointer` parameter to define the sequence. The `FileXPointer` parameter (a null pointer or a string of up to 5 bytes) is used as the beginning of the file name.

Between the time a file name is created and the file is opened, another process can create a file with the same name. Name duplication is unlikely if the other process uses these subroutines or the `mktemp` subroutine, and if the file names are chosen to avoid duplication by other means.

**Parameters**

- **String**
  Specifies the address of an array of at least the number of bytes specified by `L_tmpnam`, a constant defined in the `stdio.h` file.

  If the `String` parameter has a null value, the `tmpnam` subroutine places its result into an internal static area and returns a pointer to that area. The next call to this subroutine destroys the contents of the area.

  If the `String` parameter’s value is not null, the `tmpnam` subroutine places its results into the specified array and returns the value of the `String` parameter.
**Directory**
Points to the path name of the directory in which the file is to be created.

The `tempnam` subroutine controls the choice of a directory. If the `Directory` parameter is a null pointer or points to a string that is not a path name for an appropriate directory, the path name defined as `P_tmpdir` in the `stdio.h` file is used. If that path name is not accessible, the `/tmp` directory is used. You can bypass the selection of a path name by providing an environment variable, `TMPDIR`, in the user’s environment. The value of the `TMPDIR` environment variable is a path name for the desired temporary-file directory.

**FileXPointer**
A pointer to an initial character sequence with which the file name begins. The `FileXPointer` parameter value can be a null pointer, or it can point to a string of characters to be used as the first characters of the temporary-file name. The number of characters allowed is file system dependent, but 5 bytes is the maximum allowed.

**Return Values**
Upon completion, the `tempnam` subroutine allocates space for the string using the `malloc` subroutine, puts the generated path name in that space, and returns a pointer to the space. Otherwise, it returns a null pointer and sets the `errno` global variable to indicate the error. The pointer returned by `tempnam` may be used in the `free` subroutine when the space is no longer needed.

**Error Codes**
The `tempnam` subroutine returns the following error code if unsuccessful:

- **ENOMEM** Insufficient storage space is available.
- **EINVAL** Indicates an invalid string value.

**Related Information**
The `fopen`, `freopen`, `fdopen`, `malloc`, `free`, `realloc`, `calloc`, `malloc` or `alloca` subroutine, `malloc`, `free`, `realloc`, `calloc`, `malloc` or `alloca` subroutine, `mktemp` or `mkstemp` subroutine, `openx`, `open`, `creat` subroutine, `tmpfile` subroutine, `unlink` subroutine.

The `environment` file.

Files, Directories, and File Systems for Programmers in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Input and Output Handling Programmer’s Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

### towctrans Subroutine

**Purpose**
Character transliteration.

**Library**
Standard library (`libc.a`)

**Syntax**
```c
#include <wctype.h>
wint_t towctrans (wint_t wc, wctrans_t desc) ;
```
Description
The `towctrans` function transliterates the wide-character code `wc` using the mapping described by `desc`. The current setting of the LC_CTYPE category should be the same as during the call to `wctrans` that returned the value `desc`. If the value of `desc` is invalid (that is, not obtained by a call to `wctrans` or desc is invalidated by a subsequent call to `setlocale` that has affected category LC_CTYPE) the result is implementation-dependent.

Return Values
If successful, the `towctrans` function returns the mapped value of `wc` using the mapping described by `desc`. Otherwise it returns `wc` unchanged.

Error Codes
The `towctrans` function may fail if:

- **EINVAL**  
  desc contains an invalid transliteration descriptor.

Related Information
The `towlower` subroutine, `towupper` subroutine, `wctrans` subroutine.

The `wctype.h` file.

towlower Subroutine

Purpose
Converts an uppercase wide character to a lowercase wide character.

Library
Standard C Library (`libc.a`)

Syntax
```
#include <wchar.h>

wint_t towlower (WC);
```

Description
The `towlower` subroutine converts the uppercase wide character specified by the `WC` parameter into the corresponding lowercase wide character. The LC_CTYPE category affects the behavior of the `towlower` subroutine.

Parameters

- `WC`  
  Specifies the wide character to convert to lowercase.

Return Values
If the `WC` parameter contains an uppercase wide character that has a corresponding lowercase wide character, that wide character is returned. Otherwise, the `WC` parameter is returned unchanged.
towupper Subroutine

Purpose

Converts a lowercase wide character to an uppercase wide character.

Library

Standard C Library (libc.a)

Syntax

```c
#include <wchar.h>

wint_t towupper (WC);
```

Description

The towupper subroutine converts the lowercase wide character specified by the WC parameter into the corresponding uppercase wide character. The LC_CTYPE category affects the behavior of the towupper subroutine.

Parameters

WC Specifies the wide character to convert to uppercase.

Return Values

If the WC parameter contains a lowercase wide character that has a corresponding uppercase wide character, that wide character is returned. Otherwise, the WC parameter is returned unchanged.

Related Information

The iswalnum subroutine, iswalpha subroutine, iswcntrl subroutine, iswctype subroutine, iswdigit subroutine, iswgraph subroutine, iswlower subroutine, iswprint subroutine, iswpunct subroutine, iswspace subroutine, iswupper subroutine, iswdxigit subroutine, setlocale subroutine, towupper subroutine, wctype subroutine, wctypesubroutine, wctypesubroutine, get_wctype subroutine.

Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
**t_rcvrelldata Subroutine**

**Purpose**
Receive an orderly release indication or confirmation containing user data.

**Library**

**Syntax**

```c
#include <xti.h>

int t_rcvrelldata(
    int fd,
    struct t_discon *discon)
```

**Description**

This function is used to receive an orderly release indication for the incoming direction of data transfer and to retrieve any user data sent with the release. The argument `fd` identifies the local transport endpoint where the connection exists, and `discon` points to a `t_discon` structure containing the following members:

- `struct netbuf udata;`
- `int reason;`
- `int sequence;`

After receipt of this indication, the user may not attempt to receive more data via `t_rcv` or `t_rcvv` ("t_rcvv Subroutine" on page 416). Such an attempt will fail with `t_error` set to [TOUTSTATE]. However, the user may continue to send data over the connection if `t_sndrel` or `t_sndreldata` ("t_sndreldata Subroutine" on page 423) has not been called by the user.

The field `reason` specifies the reason for the disconnection through a protocol-dependent reason code, and `udata` identifies any user data that was sent with the disconnection; the field `sequence` is not used.

If a user does not care if there is incoming data and does not need to know the value of `reason`, `discon` may be a null pointer, and any user data associated with the disconnection will be discarded.

If `discon->udata.maxlen` is greater than zero and less than the length of the value, `t_rcvrelldata` fails with `t_errno` set to [TBUFOVFLW].

This function is an optional service of the transport provider, only supported by providers of service type `T_COTS_ORD`. The flag `T_ORDRELDATA` in the `info->flag` field returned by `t_open` or `t_getinfo` indicates that the provider supports orderly release user data; when the flag is not set, this function behaves as `t_rcvrel` and no user data is returned.

This function may not be available on all systems.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Before call</th>
<th>After call</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>x</td>
<td>/</td>
</tr>
<tr>
<td>discon-&gt;</td>
<td>udata.maxlen</td>
<td>x</td>
</tr>
<tr>
<td>discon-&gt;</td>
<td>udata.len</td>
<td>/</td>
</tr>
<tr>
<td>discon-&gt;</td>
<td>udata.buf</td>
<td>?</td>
</tr>
<tr>
<td>discon-&gt;</td>
<td>reason</td>
<td>/</td>
</tr>
<tr>
<td>discon-&gt;</td>
<td>sequence</td>
<td>/</td>
</tr>
</tbody>
</table>

**Valid States**

`T_DATAXFER, T_OUTREL`
Return Values
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and \texttt{t_errno} is set to indicate an error.

Error Codes
On failure, the \texttt{t_errno} subroutine is set to one of the following:

- \texttt{TBADF}
  The specified file descriptor does not refer to a transport endpoint.

- \texttt{TBUFOVFLW}
  The number of bytes allocated for incoming data (\texttt{maxlen}) is greater than 0 but not sufficient to store the data, and the disconnection information to be returned in \texttt{discon} will be discarded. The provider state, as seen by the user, will be changed as if the data was successfully retrieved.

- \texttt{TLOOK}
  An asynchronous event has occurred on this transport endpoint and requires immediate attention.

- \texttt{TNOREL}
  No orderly release indication currently exists on the specified transport endpoint.

- \texttt{TNOTSUPPORT}
  Orderly release is not supported by the underlying transport provider.

- \texttt{TOUTSTATE}
  The communications endpoint referenced by \texttt{fd} is not in one of the states in which a call to this function is valid.

- \texttt{TPROTO}
  This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (\texttt{t_errno}).

- \texttt{TSYSERR}
  A system error has occurred during execution of this function.

Related Information
The \texttt{t_getinfo}, \texttt{t_open}, \texttt{t_sndreldata} \textit{“t_sndreldata Subroutine” on page 423}, \texttt{t_rcvrel}, \texttt{t_sndrel} subroutines.

---

\textbf{t_rcvv Subroutine}

Purpose
Receive data or expedited data sent over a connection and put the data into one or more non-contiguous buffers.

Library
libxti.

Syntax
```c
#include <xti.h>
int t_rcvv (int fd, struct t_iovec *iov, unsigned int iovcount, int *flags) ;
```

Description
This function receives either normal or expedited data. The argument \texttt{fd} identifies the local transport endpoint through which data will arrive, \texttt{iov} points to an array of buffer address/buffer size pairs (\texttt{iov_base}, \texttt{iov_len}).
*iov_len*. The **t_rcvv** function receives data into the buffers specified by *iov[0].iov_base*, *iov[1].iov_base*, through *iov[iovcount-1].iov_base*, always filling one buffer before proceeding to the next.

**Note:** The limit on the total number of bytes available in all buffers passed (that is, *iov(0).iov_len + . . . + iov(iovcount-1).iov_len*) may be constrained by implementation limits. If no other constraint applies, it will be limited by [*INT_MAX*]. In practice, the availability of memory to an application is likely to impose a lower limit on the amount of data that can be sent or received using scatter/gather functions.

The argument *iovcount* contains the number of buffers which is limited to T_IOV_MAX (an implementation-defined value of at least 16). If the limit is exceeded, the function will fail with [TBADDATA].

The argument flags may be set on return from **t_rcvv** and specifies optional flags as described below.

By default, **t_rcvv** operates in synchronous mode and will wait for data to arrive if none is currently available. However, if O_NONBLOCK is set (via **t_open** or **fcntl**, **t_rcvv** will execute in asynchronous mode and will fail if no data is available (see [TNODATA] below).

On return from the call, if T_MORE is set in flags, this indicates that there is more data, and the current transport service data unit (TSDU) or expedited transport service data unit (ETSDU) must be received in multiple **t_rcvv** or **t_rcv** calls. In the asynchronous mode, or under unusual conditions (for example, the arrival of a signal or T_EXDATA event), the T_MORE flag may be set on return from the **t_rcvv** call even when the number of bytes received is less than the total size of all the receive buffers. Each **t_rcvv** with the T_MORE flag set indicates that another **t_rcvv** must follow to get more data for the current TSDU. The end of the TSDU is identified by the return of a **t_rcvv** call with the T_MORE flag not set. If the transport provider does not support the concept of a TSDU as indicated in the info argument on return from **t_open** or **getinfo**, the T_MORE flag is not meaningful and should be ignored. If the amount of buffer space passed in *iov* is greater than zero on the call to **t_rcvv**, then **t_rcvv** will return 0 only if the end of a TSDU is being returned to the user.

On return, the data is expedited if T_EXPEDITED is set in flags. If T_MORE is also set, it indicates that the number of expedited bytes exceeded nbytes, a signal has interrupted the call, or that an entire ETSDU was not available (only for transport protocols that support fragmentation of ETSDUs). The rest of the ETSDU will be returned by subsequent calls to **t_rcvv** which will return with T_EXPEDITED set in flags. The end of the ETSDU is identified by the return of a **t_rcvv** call with T_EXPEDITED set and T_MORE cleared. If the entire ETSDU is not available it is possible for normal data fragments to be returned between the initial and final fragments of an ETSDU.

If a signal arrives, **t_rcvv** returns, giving the user any data currently available. If no data is available, **t_rcvv** returns -1, sets **t_errno** to [TSYSERR] and errno to [EINTR]. If some data is available, **t_rcvv** returns the number of bytes received and T_MORE is set in flags.

In synchronous mode, the only way for the user to be notified of the arrival of normal or expedited data is to issue this function or check for the T_DATA or T_EXDATA events using the **t_look** function. Additionally, the process can arrange to be notified via the EM interface.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Before call</th>
<th>After call</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>X</td>
<td>/</td>
</tr>
<tr>
<td>iov</td>
<td>X/</td>
<td></td>
</tr>
<tr>
<td>iovcount</td>
<td>X</td>
<td>/</td>
</tr>
<tr>
<td>iov[0].iov_base</td>
<td>X(/)</td>
<td>=(X)</td>
</tr>
<tr>
<td>iov[0].iov_len</td>
<td>X</td>
<td>=</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Before call</th>
<th>After call</th>
</tr>
</thead>
<tbody>
<tr>
<td>iov[iovcount-1].iov_base</td>
<td>X(/)</td>
<td>=(/X)</td>
</tr>
<tr>
<td>iov[iovcount-1].iov_len</td>
<td>X</td>
<td>=</td>
</tr>
</tbody>
</table>

**Return Values**

On successful completion, t_rcvv returns the number of bytes received. Otherwise, it returns -1 on failure and t_errno is set to indicate the error.

**Error Codes**

On failure, t_errno is set to one of the following:

- **TBADDATA**: iovcount is greater than T_IOV_MAX.
- **TBADF**: The specified file descriptor does not refer to a transport endpoint.
- **TLOOK**: An asynchronous event has occurred on this transport endpoint and requires immediate attention.
- **TNODATA**: O_NONBLOCK was set, but no data is currently available from the transport provider.
- **TNOTSUPPORT**: This function is not supported by the underlying transport provider.
- **TOUTSTATE**: The communications endpoint referenced by fd is not in one of the states in which a call to this function is valid.
- **TPROTO**: This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (t_errno).
- **TSYSERR**: A system error has occurred during execution of this function.

**Related Information**

The fcntl subroutine, t_getinfo subroutine, t_look subroutine, t_open subroutine, t_rcv subroutine, t_snd subroutine, and t_sndv (“t_sndv Subroutine” on page 420) subroutine.

**t_rcvvudata Subroutine**

**Purpose**

Receive a data unit into one or more noncontiguous buffers.

**Library**

Standard library (libxti.a)

**Syntax**

```c
#include <xti.h>

int t_rcvvudata ( int fd, struct t_unitdata *unitdata, struct t_iovec *iov, unsigned int iovcount, int *flags )
```

**Description**

This function is used in connectionless mode to receive a data unit from another transport user. The argument fd identifies the local transport endpoint through which data will be received, unitdata holds information associated with the received data unit, iovcount contains the number of non-contiguous udata buffers which is limited to T_IOV_MAX (an implementation-defined value of at least 16), and flags is set on return to indicate that the complete data unit was not received. If the limit on iovcount is exceeded, the function fails with [TBADDATA]. The argument unitdata points to a t_unitdata structure containing the following members:

- struct netbuf addr;
- struct netbuf opt;
- struct netbuf udata;
The `maxlen` field of `addr` and `opt` must be set before calling this function to indicate the maximum size of the buffer for each. The `udata` field of `t_unitdata` is not used. The `iov_len` and `iov_base` fields of `iov[0]` through `iov[iovcount-1]` must be set before calling `t_rcvvudata` to define the buffer where the user data will be placed. If the maxlen field of `addr` or `opt` is set to zero then no information is returned in the `buf` field for this parameter.

On return from this call, `addr` specifies the protocol address of the sending user, `opt` identifies options that were associated with this data unit, and `iov[0].iov_base` through `iov[iovcount-1].iov_base` contains the user data that was received. The return value of `t_rcvvudata` is the number of bytes of user data given to the user.

**Note:** The limit on the total number of bytes available in all buffers passed (that is, `iov(0).iov_len + ... + iov(iovcount-1).iov_len`) may be constrained by implementation limits. If no other constraint applies, it will be limited by `INT_MAX`. In practice, the availability of memory to an application is likely to impose a lower limit on the amount of data that can be sent or received using scatter/gather functions.

By default, `t_rcvvudata` operates in synchronous mode and waits for a data unit to arrive if none is currently available. However, if `O_NONBLOCK` is set (via `t_open` or `fcntl`), `t_rcvvudata` executes in asynchronous mode and fails if no data units are available. If the buffers defined in the `iov[]` array are not large enough to hold the current data unit, the buffers will be filled and T_MORE will be set in flags on return to indicate that another `t_rcvvudata` should be called to retrieve the rest of the data unit. Subsequent calls to `t_rcvvudata` will return zero for the length of the address and options, until the full data unit has been received.

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Before call</th>
<th>After call</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fd</code></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><code>unitdata-&gt;addr.maxlen</code></td>
<td>X</td>
<td>=</td>
</tr>
<tr>
<td><code>unitdata-&gt;addr.len</code></td>
<td>/</td>
<td>X</td>
</tr>
<tr>
<td><code>unitdata-&gt;addr.buf</code></td>
<td>?(/)</td>
<td>=(/)</td>
</tr>
<tr>
<td><code>unitdata-&gt;opt.maxlen</code></td>
<td>X</td>
<td>=</td>
</tr>
<tr>
<td><code>unitdata-&gt;opt.len</code></td>
<td>/</td>
<td>X</td>
</tr>
<tr>
<td><code>unitdata-&gt;opt.buf</code></td>
<td>?(/)</td>
<td>=(?</td>
</tr>
<tr>
<td><code>unitdata-&gt;udata.maxlen</code></td>
<td>/</td>
<td>=</td>
</tr>
<tr>
<td><code>unitdata-&gt;udata.len</code></td>
<td>/</td>
<td>=</td>
</tr>
<tr>
<td><code>unitdata-&gt;udata.buf</code></td>
<td>/</td>
<td>=</td>
</tr>
<tr>
<td><code>iov[0].iov_base</code></td>
<td>X</td>
<td>=(X)</td>
</tr>
<tr>
<td><code>iov[0].iov_len</code></td>
<td>X</td>
<td>=</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>iov[iovcount-1].iov_base</code></td>
<td>X(/)</td>
<td>=(X)</td>
</tr>
<tr>
<td><code>iov[iovcount-1].iov_len</code></td>
<td>X</td>
<td>=</td>
</tr>
<tr>
<td><code>iovcount</code></td>
<td>X</td>
<td>/</td>
</tr>
<tr>
<td><code>flags</code></td>
<td>/</td>
<td>/</td>
</tr>
</tbody>
</table>

### Return Values

On successful completion, `t_rcvvudata` returns the number of bytes received. Otherwise, it returns -1 on failure and `t_errno` is set to indicate the error.
Error Codes
On failure, t_errno is set to one of the following:

- **TBADDATA**
  - iovcount is greater than T_IOV_MAX.
- **TBAFD**
  - The specified file descriptor does not refer to a transport endpoint.
- **TBUFOVFLW**
  - The number of bytes allocated for the incoming protocol address or options (maxlen) is greater than 0 but not sufficient to store the information. The unit data information to be returned in unitdata will be discarded.
- **TLOOK**
  - An asynchronous event has occurred on this transport endpoint and requires immediate attention.
- **TNODATA**
  - O_NONBLOCK was set, but no data units are currently available from the transport provider.
- **TNOTSUPPORT**
  - This function is not supported by the underlying transport provider.
- **TOUTSTATE**
  - The communications endpoint referenced by fd is not in one of the states in which a call to this function is valid.
- **TPROTO**
  - This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (t_errno).
- **TSYSERR**
  - A system error has occurred during execution of this function.

Related Information
The [fcntl subroutine](#), [t_alloc subroutine](#), [t_open subroutine](#), [t_rcvudata subroutine](#), [t_rcvuderr subroutine](#), [t_sndudata subroutine](#), [t_sndvudata subroutine](#) subroutine.

### t_sndv Subroutine

**Purpose**
Send data or expedited data, from one or more non-contiguous buffers, on a connection.

**Library**
Standard library ([libxti.a](#))

**Syntax**
```c
#include <xti.h>

int t_sndv (int fd, const struct t_iovec *iov, unsigned int iovcount, int flags)
```

**Description**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Before call</th>
<th>After call</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>X</td>
<td>/</td>
</tr>
<tr>
<td>iov</td>
<td>X</td>
<td>/</td>
</tr>
<tr>
<td>iovcount</td>
<td>X</td>
<td>/</td>
</tr>
<tr>
<td>iov[0].iov_base</td>
<td>X(X)</td>
<td>/</td>
</tr>
<tr>
<td>iov[0].iov_len</td>
<td>X</td>
<td>/</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iov[iovcount-1].iov_base</td>
<td>X(X)</td>
<td>/</td>
</tr>
<tr>
<td>iov[iovcount-1].iov_len</td>
<td>X</td>
<td>=</td>
</tr>
<tr>
<td>flags</td>
<td>X</td>
<td>/</td>
</tr>
</tbody>
</table>

This function is used to send either normal or expedited data. The argument fd identifies the local transport endpoint over which data should be sent, iov points to an array of buffer address/length.
pairs. `t_sndv` sends data contained in buffers `iov[0]`, `iov[1]`, through `iov[iovcount-1]`. `iovcount` contains the number of non-contiguous data buffers which is limited to `T_IOV_MAX` (an implementation-defined value of at least 16). If the limit is exceeded, the function fails with [TBADDATA].

**Note:** The limit on the total number of bytes available in all buffers passed (that is: `iov(0).iov_len + . . . + iov(iovcount-1).iov_len`) may be constrained by implementation limits. If no other constraint applies, it will be limited by `INT_MAX`. In practice, the availability of memory to an application is likely to impose a lower limit on the amount of data that can be sent or received using scatter/gather functions.

The argument `flags` specifies any optional flags described below:

### T_EXPEDITED
If set in `flags`, the data will be sent as expedited data and will be subject to the interpretations of the transport provider.

### T_MORE
If set in `flags`, this indicates to the transport provider that the transport service data unit (TSDU) (or expedited transport service data unit ETSDU) is being sent through multiple `t_sndv` calls. Each `t_sndv` with the T_MORE flag set indicates that another `t_sndv` (or `t_snd`) will follow with more data for the current TSDU (or ETSDU).

The end of the TSDU (or ETSDU) is identified by a `t_sndv` call with the T_MORE flag not set. Use of T_MORE enables a user to break up large logical data units without losing the boundaries of those units at the other end of the connection. The flag implies nothing about how the data is packaged for transfer below the transport interface. If the transport provider does not support the concept of a TSDU as indicated in the `info` argument on return from `t_open` or `getinfo`, the T_MORE flag is not meaningful and will be ignored if set. The sending of a zero-length fragment of a TSDU or ETSDU is only permitted where this is used to indicate the end of a TSDU or ETSDU, that is, when the T_MORE flag is not set. Some transport providers also forbid zero-length TSDUs and ETSDUs. See Appendix A for a fuller explanation.

If set in `flags`, requests that the provider transmit all data that it has accumulated but not sent. The request is a local action on the provider and does not affect any similarly named protocol flag (for example, the TCP PUSH flag). This effect of setting this flag is protocol-dependent, and it may be ignored entirely by transport providers which do not support the use of this feature.

**Note:** The communications provider is free to collect data in a send buffer until it accumulates a sufficient amount for transmission.

By default, `t_sndv` operates in synchronous mode and may wait if flow control restrictions prevent the data from being accepted by the local transport provider at the time the call is made. However, if O_NONBLOCK is set (via `t_open` or `fcntl`), `t_sndv` executes in asynchronous mode, and will fail immediately if there are flow control restrictions. The process can arrange to be informed when the flow control restrictions are cleared via either `t_look` or the EM interface.

On successful completion, `t_sndv` returns the number of bytes accepted by the transport provider. Normally this will equal the total number of bytes to be sent, that is, `iov[0].iov_len + . . . + iov[iovcount-1].iov_len`

However, the interface is constrained to send at most `INT_MAX` bytes in a single send. When `t_sndv` has submitted `INT_MAX` (or lower constrained value, see the note above) bytes to the provider for a single call, this value is returned to the user. However, if O_NONBLOCK is set or the function is interrupted by a signal, it is possible that only part of the data has actually been accepted by the communications provider.
In this case, \texttt{t_sndv} returns a value that is less than the value of \texttt{nbytes}. If \texttt{t_sndv} is interrupted by a signal before it could transfer data to the communications provider, it returns \texttt{-1} with \texttt{t_errno} set to \texttt{[TSYSERR]} and \texttt{errno} set to \texttt{[EINTR]}

If the number of bytes of data in the \texttt{iov} array is zero and sending of zero octets is not supported by the underlying transport service, \texttt{t_sndv} returns \texttt{-1} with \texttt{t_errno} set to \texttt{[TBADDATA]}.

The size of each TSDU or ETSDU must not exceed the limits of the transport provider as specified by the current values in the TSDU or ETSDU fields in the \texttt{info} argument returned by \texttt{t_getinfo}.

The error \texttt{[TLOOK]} is returned for asynchronous events. It is required only for an incoming disconnect event but may be returned for other events.

**Return Values**

On successful completion, \texttt{t_sndv} returns the number of bytes accepted by the transport provider. Otherwise, \texttt{-1} is returned on failure and \texttt{t_errno} is set to indicate the error.

**Notes:**

1. In synchronous mode, if more than \texttt{INT_MAX} bytes of data are passed in the \texttt{iov} array, only the first \texttt{INT_MAX} bytes will be passed to the provider.
2. If the number of bytes accepted by the communications provider is less than the number of bytes requested, this may either indicate that \texttt{O_NONBLOCK} is set and the communications provider is blocked due to flow control, or that \texttt{O_NONBLOCK} is clear and the function was interrupted by a signal.

**Error Codes**

On failure, \texttt{t_errno} is set to one of the following:

- \texttt{TBADDATA} Illegal amount of data:
  - A single send was attempted specifying a TSDU (ETSDU) or fragment TSDU (ETSDU) greater than that specified by the current values of the TSDU or ETSDU fields in the \texttt{info} argument.
  - A send of a zero byte TSDU (ETSDU) or zero byte fragment of a TSDU (ETSDU) is not supported by the provider.
  - Multiple sends were attempted resulting in a TSDU (ETSDU) larger than that specified by the current value of the TSDU or ETSDU fields in the \texttt{info} argument the ability of an XTI implementation to detect such an error case is implementation-dependent (see CAVEATS, below).
  - \texttt{iovcount} is greater than \texttt{T_IOV_MAX}.

- \texttt{TBADF} The specified file descriptor does not refer to a transport endpoint.
- \texttt{TBADFLAG} An invalid flag was specified.
- \texttt{TFLOW} \texttt{O_NONBLOCK} was set, but the flow control mechanism prevented the transport provider from accepting any data at this time.
- \texttt{TLOOK} An asynchronous event has occurred on this transport endpoint.
- \texttt{TNOTSUPPORT} This function is not supported by the underlying transport provider.
- \texttt{TOUTSTATE} The communications endpoint referenced by \texttt{fd} is not in one of the states in which a call to this function is valid.
- \texttt{TPROTO} This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (\texttt{t_errno}).
- \texttt{TSYSERR} A system error has occurred during execution of this function.

**Related Information**

The \texttt{t_getinfo} subroutine, \texttt{t_open} subroutine, \texttt{t_rcvv} subroutine, \texttt{t_rcv} subroutine, \texttt{t_snd} subroutine.
t_sndreldata Subroutine

Purpose
Initiate/respond to an orderly release with user data.

Library

Syntax
#include <xti.h>

int t_sndreldata(int fd, struct t_discon *discon)

Description
This function is used to initiate an orderly release of the outgoing direction of data transfer and to send user data with the release. The argument fd identifies the local transport endpoint where the connection exists, and discon points to a t_discon structure containing the following members:

```c
struct netbuf udata;
int reason;
int sequence;
```

After calling t_sndreldata, the user may not send any more data over the connection. However, a user may continue to receive data if an orderly release indication has not been received.

The field reason specifies the reason for the disconnection through a protocol-dependent reason code, and udata identifies any user data that is sent with the disconnection; the field sequence is not used.

The udata structure specifies the user data to be sent to the remote user. The amount of user data must not exceed the limits supported by the transport provider, as returned in the discon field of the info argument of t_open or t_getinfo. If the len field of udata is zero or if the provider did not return T_ORDRELDATA in the t_open flags, no data will be sent to the remote user.

If a user does not wish to send data and reason code to the remote user, the value of discon may be a null pointer.

This function is an optional service of the transport provider, only supported by providers of service type T_COTS_ORD. The flag T_ORDRELDATA in the info->flag field returned by t_open or t_getinfo indicates that the provider supports orderly release user data; when the flag is not set, this function behaves as t_rcvrel and no user data is returned.

This function may not be available on all systems.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Before call</th>
<th>After call</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>x</td>
<td>/</td>
</tr>
<tr>
<td>discon-&gt;</td>
<td>udata.maxlen</td>
<td>/</td>
</tr>
<tr>
<td>discon-&gt;</td>
<td>udata.len</td>
<td>x</td>
</tr>
<tr>
<td>discon-&gt;</td>
<td>udata.buf</td>
<td>?(?)</td>
</tr>
<tr>
<td>discon-&gt;</td>
<td>reason</td>
<td>?</td>
</tr>
<tr>
<td>discon-&gt;</td>
<td>sequence</td>
<td>/</td>
</tr>
</tbody>
</table>

Valid States
T_DATAXFER, T_INREL
Error Codes
On failure, t_errno is set to one of the following:

[TBADDATA]
   The amount of user data specified was not within the bounds allowed by the transport provider, or
   user data was supplied and the provider did not return T_ORDRELDATA in the t_open flags.

[TBADF]
   The specified file descriptor does not refer to a transport endpoint.

[TFLOW]
   O_NONBLOCK was set, but the flow control mechanism prevented the transport provider from
   accepting the function at this time.

[TLOOK]
   An asynchronous event has occurred on this transport endpoint and requires immediate attention.

[TNOTSUPPORT]
   Orderly release is not supported by the underlying transport provider.

[TOUTSTATE]
   The communications endpoint referenced by fd is not in one of the states in which a call to this
   function is valid.

[PROTO]
   This error indicates that a communication problem has been detected between XTI and the
   transport provider for which there is no other suitable XTI error (t_errno).

[TSYSERR]
   A system error has occurred during execution of this function.

Return Value
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and t_errno is
set to indicate an error.

Related Information
The t_getinfo, t_open, t_rcvreldata ("t_rcvreldata Subroutine" on page 415), t_rcvrel and t_sndrel
subroutines.

_t_sndvudata Subroutine

Purpose
Send a data unit from one or more noncontiguous buffers.

Library

Syntax

#include <xti.h>

int t_sndvudata(
   int fd,
   struct t_unitdata *unitdata,
   struct t_iovec *iov,
   unsigned int iovcount)
Description

This function is used in connectionless mode to send a data unit to another transport user. The argument
*fd* identifies the local transport endpoint through which data will be sent, *iovcount* contains the number of
non-contiguous udata buffers and is limited to an implementation-defined value given by T_IOV_MAX,
which is at least 16, and *unitdata* points to a *t_unitdata* structure containing the following members:

```c
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
```

If the limit on *iovcount* is exceeded, the function fails with [TBADDATA].

In *unitdata*, *addr* specifies the protocol address of the destination user, and *opt* identifies options that the
user wants associated with this request. The *udata* field is not used. The user may choose not to specify
what protocol options are associated with the transfer by setting the *len* field of *opt* to zero. In this case,
the provider may use default options.

The data to be sent is identified by *iov[0]* through *iov[iovcount-1]*.

```c
The limit on the total number of bytes available in all buffers passed (that is:

`iov(0).iov_len + ... + iov(iovcount-1).iov_len`)
```

may be constrained by implementation limits. If no other constraint applies, it will be limited by
[INT_MAX]. In practice, the availability of memory to an application is likely to impose a lower limit on
the amount of data that can be sent or received using scatter/gather functions.

By default, *t_sndvudata* operates in synchronous mode and may wait if flow control restrictions prevent
the data from being accepted by the local transport provider at the time the call is made. However, if
O_NONBLOCK is set (via *t_open* or *fcntl*), *t_sndvudata* executes in asynchronous mode and will fail
under such conditions. The process can arrange to be notified of the clearance of a flow control restriction
via either *t_look* or the EM interface.

If the amount of data specified in *iov[0]* through *iov[iovcount-1]* exceeds the TSDU size as returned in
the *tsdu* field of the *info* argument of *t_open* or *t_getinfo* or is zero and sending of zero octets is not
supported by the underlying transport service, a [TBADDATA] error is generated. If *t_sndvudata* is called
before the destination user has activated its transport endpoint (see *t_bind*), the data unit may be
discarded.

If it is not possible for the transport provider to immediately detect the conditions that cause the errors
[TBADDADDR] and [TBADOPT], these errors will alternatively be returned by *t_rcvuderr*. An application
must therefore be prepared to receive these errors in both of these ways.

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Before call</th>
<th>After call</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>fd</em></td>
<td>x</td>
<td>/</td>
</tr>
<tr>
<td><em>unitdata-&gt;addr</em></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td><em>unitdata-&gt;addr</em></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td><em>unitdata-&gt;addr</em></td>
<td>x(x)</td>
<td>x</td>
</tr>
<tr>
<td><em>unitdata-&gt;opt</em></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td><em>unitdata-&gt;opt</em></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td><em>unitdata-&gt;opt</em></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td><em>unitdata-&gt;opt</em></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td><em>unitdata-&gt;udata</em></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td><em>unitdata-&gt;udata</em></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td><em>unitdata-&gt;udata</em></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td><em>iov[0].iov_base</em></td>
<td>x(x)</td>
<td>(=)</td>
</tr>
<tr>
<td>left&gt;iov[0].iov_len</td>
<td>x</td>
<td>(=)</td>
</tr>
</tbody>
</table>
Parameters
iov[iovcount-1].iov_base
iov[iovcount-1].iov_len
iovcount

Before call
x(x)
x

After call
(=)
=
/

Valid States
T_IDLE

Error Codes
On failure, t_errno is set to one of the following:

[TBADADDR] The specified protocol address was in an incorrect format or contained illegal information.

[TBADATA] Illegal amount of data.

- A single send was attempted specifying a TSDU greater than that specified in the info argument, or a send of a zero byte TSDU is not supported by the provider.

- iovcount is greater than T_IOV_MAX.

[TBADF] The specified file descriptor does not refer to a transport endpoint.

[TBADOPT] The specified options were in an incorrect format or contained illegal information.

[TFLOW] O_NONBLOCK was set, but the flow control mechanism prevented the transport provider from accepting any data at this time.

[TLOOK] An asynchronous event has occurred on this transport endpoint.

[TNOTSUPPORT] The communications endpoint referenced by fd is not in one of the states in which a call to this function is valid.

[TPROTO] This error indicates that a communication problem has been detected between XTI and the transport provider for which there is no other suitable XTI error (t_errno).

[TSYSERR] A system error has occurred during execution of this function.

Return Values
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and t_errno is set to indicate an error.

Related Information
The fcntl, t_alloc, t_open, t_rcvudata, t_rcvvudata, t_sndudata subroutines.

t_sysconf Subroutine

Purpose
Get configurable XTI variables.

Library
Standard library (libxti.a)

Syntax
#include <xti.h>

int t_sysconf ( int name)
Description

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Before call</th>
<th>After call</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>X</td>
<td>/</td>
</tr>
</tbody>
</table>

The `t_sysconf` function provides a method for the application to determine the current value of configurable and implementation-dependent XTI limits or options.

The `name` argument represents the XTI system variable to be queried. The following table lists the minimal set of XTI system variables from `xti.h` that can be returned by `t_sysconf`, and the symbolic constants, defined in `xti.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>T_IOV_MAX</td>
<td>_SC_T_IOV_MAX</td>
</tr>
</tbody>
</table>

Return Values

If `name` is valid, `t_sysconf` returns the value of the requested limit/option (which might be -1) and leaves `t_errno` unchanged. Otherwise, a value of -1 is returned and `t_errno` is set to indicate an error.

Error Codes

On failure, `t_errno` is set to the following:

- **TBADFLAG**: `name` has an invalid value.

Related Information

The `t_rcvv` ("t_rcvv Subroutine" on page 416) subroutine, `t_rcvvudata` ("t_rcvvudata Subroutine" on page 418) subroutine, `t_sndv` ("t_sndv Subroutine" on page 420) subroutine, `t_sndvudata` ("t_sndvudata Subroutine" on page 424) subroutine.

**trc_close Subroutine**

Purpose

Closes and frees a trace log object.

Library

`libtrace.a`

Syntax

```
#include <sys/libtrace.h>

int trc_close (trc_log_handle_t handle);
```

Description

The `trc_close` subroutine closes a trace log object. The object must have been opened with the `trc_open` subroutine. If the `TRC_RETAIN_HANDLE` type was specified at open time, the `trc_close` subroutine must be called after a call to the `trc_open` subroutine, regardless of whether the open succeeded or not.
Parameters

(handle) Contains the handle returned from a successful call to the trc_open subroutine.

Return Values

Upon successful completion, the trc_close subroutine returns a 0.

Error Codes

Upon error, the trc_close subroutine sets the errno global variable and returns the error from the fclose subroutine. In addition, EINVAL is returned if handle contains an invalid trc_log_handle_t object.

Related Information

"trc_open Subroutine" on page 439, "trc_read Subroutine" on page 442, "trc_loginfo Subroutine" on page 437, "trc_find_first, trc_find_next, and trc_compare Subroutine, trc_seek and trc_tell Subroutine" on page 446, "trc_libcntl Subroutine" on page 435, "trc_strerror Subroutine" on page 447, "trc_perror Subroutine" on page 441, "trcstart Subroutine" on page 453, "trcstop Subroutine" on page 454, and "trcoff Subroutine" on page 452.

The trace daemon in AIX 5L Version 5.3 Commands Reference, Volume 5.

The trcrt, trcstop, and trcupdate commands in AIX 5L Version 5.3 Commands Reference, Volume 5.

trc_find_first, trc_find_next, and trc_compare Subroutine

Purpose

Finds the first, or next, occurrence of the argument, or compares the current entry with the argument.

Library

libtrace.a

Syntax

#include <sys/libtrace.h>

int trc_find_first (handle, argp, ret)
trc_log_handle_t handle;
trc_logsearch_t *argp;
trc_read_t *ret;

int trc_find_next (handle, argp, ret)
trc_log_handle_t handle;
trc_logsearch_t *argp;
trc_read_t *ret;

int trc_compare (handle, argp)
trc_log_handle_t handle;
trc_logsearch_t *argp;

Description

The trc_find_first subroutine finds the first occurrence of the trace log entry matching the argument pointed to by the argp parameter. The trc_find_next subroutine finds the next occurrence of the argument starting from the current position in the log object. If the search argument pointer, argp, is NULL, the argument from the previous search is used. Both the trc_find_first and trc_find_next subroutines return the item found.
The **trc_compare** subroutine is used to check the current entry against the argument. No data is read. It is useful when implementing exit criteria, where you need to find entries according to some criteria, but then check for an exit criteria which is not part of the normal search.

**Parameters**

- **handle**: Contains the handle returned from a successful call to the **trc_open** subroutine.
- **argp**: Points to the argument list as defined in the `/usr/include/sys/libtrace.a` file. Arguments may be chained together to perform complex searches.
- **ret**: Points to the **trc_read_t** structure to be returned. The **trc_free** subroutine should be used to free data referenced from the **trc_read_t** data type, unless **TRC_LOGLIVE** was specified at open time.

The search argument consists of three parts, the operator, **tls_op**, and the left and right sides.

The operator values can be easily identified, because they have the form **TLS_OP_.....**. Operators are split into two categories, leaf and compound operators. Leaf operators are operators that compare the field on the left with the value on the right. Compound operators are used to compare two expressions, (for example) to combined expressions.

Leaf operations may be performed using numeric or string data. If performed on string data, the **strcmp** libc string compare function is used to do the comparison for all operators except **TLS_OP_SUBSTR**. The valid leaf operators are:

- **TLS_OP_EQUAL**: Exactly equal
- **TLS_OP_NE**: Not equal
- **TLS_OP_LT**: Less than
- **TLS_OP_LE**: Less than or equal
- **TLS_OP_GT**: Greater than
- **TLS_OP_GE**: Greater than or equal
- **TLS_OP_SUBSTR**: The string on the left contains the string on the right.

The compound operators are:

- **TLS_OP_AND**: The logical AND of the results of the left and right expressions.
- **TLS_OP_OR**: The logical OR of the results of the left and right expressions.
- **TLS_OP_XOR**: The exclusive or of the results of the left and right expressions.
- **TLS_OP_NOT**: The negation of the argument referenced by **tls_left**.

The left and right sides of the expression are defined as follows:

- **tls_left** and **tls_right**: These are used when the operator requires the left and right sides to be an expression, (for example) when it is a compound operator. **tls_left** and **tls_right** point to other **trc_logsearch_t** structures.
- **tls_field** and corresponding values: For a leaf operation, **tls_field**, on the left, specifies the field to be compared. The field names can be identified easily, because they all have the form **TLS_MATCH_.....**. The righthand side is a value specified according to the data type of the field on the left.
The following table shows the lefthand field values and their corresponding righthand side data values:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS_MATCH_HOOKID</td>
<td>tls_shortvalue</td>
<td>Compare the hookid with a short data item.</td>
</tr>
<tr>
<td>TLS_MATCH_HOOK_AND_SUBHOOK</td>
<td>tls_intvalue</td>
<td>Compare the hook and subhook, 28 bits, with the specified integer. Note that the field is of the form 0x0hhhsss, where hhh is the hook id, and ssss is the subhook.</td>
</tr>
<tr>
<td>TLS_MATCH_HOOKSET</td>
<td>tls_hooksetvalue</td>
<td>The bit map specifying the hooks to be tested for. This allows you to test for multiple hooks with one search argument. The bit map is manipulated with the trc_hkemptysset, trc_hkfillset, trc_hkaddset, and trc_hkdelset subroutines.</td>
</tr>
<tr>
<td>TLS_MATCH_TIME</td>
<td>tls_longvalue</td>
<td>Compare the time value in nanoseconds from the start of the trace.</td>
</tr>
<tr>
<td>TLS_MATCH_TID</td>
<td>tls_longvalue</td>
<td>Thread id</td>
</tr>
<tr>
<td>TLS_MATCH_PID</td>
<td>tls_longvalue</td>
<td>Process id</td>
</tr>
<tr>
<td>TLS_MATCH_RAWOFFSET</td>
<td>tls_longvalue</td>
<td>Raw file offset</td>
</tr>
<tr>
<td>TLS_MATCH_CPUID</td>
<td>tls_intvalue</td>
<td>cpu id</td>
</tr>
<tr>
<td>TLS_MATCH_RCPU</td>
<td>tls_intvalue</td>
<td>Remaining cpus in the trace.</td>
</tr>
<tr>
<td>TLS_MATCH_FLAGS</td>
<td>tls_intvalue</td>
<td>Compare with trcr_flags</td>
</tr>
<tr>
<td>TLS_MATCH_INTR_DEPTH</td>
<td>tls_intvalue</td>
<td>Compare with trchi_intr_depth</td>
</tr>
<tr>
<td>TLS_MATCH_PROCNAME</td>
<td>tls_strvalue</td>
<td>Process name</td>
</tr>
<tr>
<td>TLS_MATCH_SVCNAME</td>
<td>tls_strvalue</td>
<td>svc name</td>
</tr>
<tr>
<td>TLS_MATCH_PRI</td>
<td>tls_intvalue</td>
<td>Dispatch priority</td>
</tr>
<tr>
<td>TLS_MATCH_TICKS</td>
<td>tls_longvalue</td>
<td>Match with the number of timer register ticks since the start of the trace.</td>
</tr>
<tr>
<td>TLS_MATCH_DATA</td>
<td>tls_strvalue</td>
<td>Compare string with the ascii data, trchi_ascii</td>
</tr>
<tr>
<td>TLS_MATCH_FILENAME</td>
<td>tls_strvalue</td>
<td>Compare with trchi_filename</td>
</tr>
<tr>
<td>TLS_MATCH_TRCONTIME</td>
<td>tls_longvalue</td>
<td>Compare with trchi_trcontime</td>
</tr>
<tr>
<td>TLS_MATCH_TRCOFFTIME</td>
<td>tls_longvalue</td>
<td>Compare with trchi_trcofftime</td>
</tr>
</tbody>
</table>

**Return Values**

Upon successful completion, the trc_find_first, trc_find_next, and trc_compare subroutines return 0.

**Error Codes**

Upon error, the errno global variable is set to a value from the errno.h file. The trc_find_first, trc_find_next, and trc_compare subroutines return either a value from the errno.h file, or an error value from the libtrace.h file.
EINVAL The handle is invalid, or the search argument is invalid.
TRCE_EOF No matching item was found, or no more matching items exist. The errno global variable is set to 0.
TRCE_BADFORMAT The log object contains badly formatted data. The errno global variable is set to EINVAL.

Examples
1. Find the SVC hooks, 101 and 104, for program mypgm.
   
   ```c
   int rv;
   trc_loghandle_t h;
   trc_read_t r;
   trc_logsearch_t t1, t2, t3, t4, t5;
   
   /* Setup the leaf search arguments. */
   t1.tls_op = TLS_OP_EQUAL;
   t1.tls_field = TLS_MATCH_HOOKID;
   t1.tls_shortvalue = 0x101;
   t2.tls_op = TLS_OP_EQUAL;
   t2.tls_field = TLS_MATCH_HOOKID;
   t2.tls_shortvalue = 0x104;
   t3.tls_op = TLS_OP_EQUAL;
   t3.tls_field = TLS_MATCH_PROCNAME;
   t3.tls_strvalue = "mypgm";
   
   /* Join the items and form a single search tree. */
   t4.tls_op = TLS_OP_AND;
   t4.tls_left = &t1
   t4.tls_right = &t2
   t5.tls_op = TLS_OP_AND;
   t5.tls_left = &t3
   t5.tls_right = &t4
   
   /* Open the default trace log object. */
   rv = trc_open("", ", TRC_LOGREAD|TRC_LOGPROC, &h);
   if (rv) {
      trc_perror(h, rv, "open");
      return(rv);
   }
   
   /* Do the search. */
   rv = trc_find_first(h, &t5, &r);
   if (rv) {
      trc_perror(h, rv, "find test");
      return(rv);
   }
   
   ...  
   ```

   Note that subsequent entries matching this search could be returned with the following:
   
   ```c
   rv = trc_find_next(h, NULL, &r);
   ```

   After a find, trc_find_next can be used to change the search argument without starting the search over. In other words, trc_find_first always starts from the beginning of the file, while trc_find_next starts from the current position in the file, but either one can change the search argument.

2. Find the SVC hooks, 101 and 104, for program mypgm. Use a single argument to search for both hook ids.
   
   ```c
   int rv;
   trc_loghandle_t h;
   trc_read_t r;
   trc_logsearch_t t1, t2, t3;
   trc_hookset_t hs;
   
   /* Setup the hook set. */
   trc_hkemptyset(hs);
   ```
(void)trc_hkaddset(hs, 0x101);
(void)trc_hkaddset(hs, 0x104);

/* Setup the leaf search arguments. */
t1.tls_op = TLS_OP_EQUAL;
t1.tls_field = TLS_MATCH_HOOKSET;
t1.tls_hooksetvalue = hs;
t2.tls_op = TLS_OP_EQUAL;
t2.tls_field = TLS_MATCH_PROCNAME;
t2.tls_strvalue = "mypgm";
/* Join the items and form a single search tree. */
t3.tls_op = TLS_OP_AND;
t3.tls_left = &t1;
t3.tls_right = &t2;
/* Open the default trace log object. */
rv = trc_open("", ", TRC_LOGREAD|TRC_LOGPROC, &h);
if (rv)
  
  trc_perror(h, rv, "open");
  return(rv);

/* Do the search. */
rv = trc_find_first(h, &t3, &r);
if (rv)
  
  trc_perror(h, rv, "find test");
  return(rv);

...

Related Information

"trc_open Subroutine" on page 439, "trc_close Subroutine" on page 427, "trc_read Subroutine" on page 442, "trc_loginfo Subroutine" on page 437, "trc_seek and trc_tell Subroutine" on page 446, "trc_libcntl Subroutine" on page 435, "trc_strerror Subroutine" on page 447, "trc_perror Subroutine" on page 441, "trcstart Subroutine" on page 453, "trcstop Subroutine" on page 454, and "trc_hkemtpyset, trc_hkfillset, trc_hkaddset, trc_hkdelset, and trc_hkisset Subroutine" on page 433.

The trace daemon in AIX 5L Version 5.3 Commands Reference, Volume 5.

The trc_f, trcstop and trcupdate commands in AIX 5L Version 5.3 Commands Reference, Volume 5.

---

trc_free Subroutine

Purpose
Frees memory allocated by the trc_read, trc_find, trc_loginfo, or trc_hookname subroutine.

Library
libtrace.a

Syntax
#include <sys/libtrace.h>

int trc_free (param);
void *param;
Description
The trc_free subroutine is used to free memory associated with data structures returned by the trace retrieval API. It does not free the storage for the base structure, however, only storage allocated by the API on behalf of the user. The pointer must point to one of the following:

trc_read_t
    Data returned by the trc_read or trc_find subroutine.

trc_loginfo_t
    Data returned by the trc_loginfo subroutine.

trc_hookname_t
    Data returned by the trc_hookname subroutine.

trc_logpos_t
    A log position object returned by the trc_tell subroutine.

A log handle, trc_loghandle_t, must be freed using the trc_close subroutine.

For example, trc_free(&trc_data), where trc_data is of type trc_read_t, frees the storage referenced by the trc_data structure, but does not free trc_data since it must be pre-allocated by the user.

Parameters
parmp
    Points to a structure as described above.

Return Values
Upon successful completion, the trc_free subroutine returns 0.

Error Codes
EINVAL
    The parmp parameter points to an unsupported data type.

Related Information
trc_read Subroutine on page 442, trc_loginfo Subroutine on page 437, trc_find_first, trc_find_next, and trc_compare Subroutine on page 428, trc_hookname Subroutine on page 434, trc_seek and trc_tell Subroutine on page 446, trc_strerror Subroutine on page 447, and trc_perror Subroutine on page 441.

trc_hkemptyset, trc_hkfillset, trc_hkaddset, trc_hkdelset, and trc_hkisset Subroutine

Purpose
Manipulates a trace hook set.

Library
libtrace.a

Syntax
#include <sys/libtrace.h>

void trc_hkemptyset(hookset)
trc_hookset_t hookset;
void trc_hkfillset(hookset)
  trc_hookset_t hookset;

int trc_hkaddset(hookset, hook)
  trc_hookset_t hookset;
  short hook;

int trc_hkdelset(hookset, hook)
  trc_hookset_t hookset;
  short hook;

int trc_hkisset (hookset, hook)
  trc_hookset_t hookset;
  short hook

Description
These subroutines manipulate a trace hook set used by the trc_find subroutines. This hook set can be used to search for several trace hooks simultaneously.

Parameters

hookset References the hook set to be operated on.

hook Specifies a hook value in the range 0x000 - 0xffff.

Return Values
The trc_hkaddset, trc_hkdelset, and trc_hkisset subroutines return EINVAL if the hook is out of range (that is, greater than 0xffff).

The trc_hkaddset subroutine returns 0 if the hook wasn’t in the set, and -1 if it was already present.

The trc_hkdelset subroutine returns 0 if the hook was in the set, and -1 if it wasn’t present.

The trc_hkisset subroutine returns 0 if the hook isn’t present, and -1 if it is present.

Related Information
“trc_loginfo Subroutine” on page 437 and “trc_find_first, trc_find_next, and trc_compare Subroutine” on page 428.

trc_hookname Subroutine

Purpose
Returns one or all hooks and associated names from the template file.

Library
libtrace.a

Syntax
#include <sys/libtrace.h>

int trc_hookname (handle, hook, hooknamep)
  trc_log_handle_t handle;
  trc_hookid_t hook;
  trc_hookname_t *hooknamep;
Description
The `trc_hookname` subroutine returns one or more hook ids and their associated descriptions. This allows a trace data formatter to provide a hook selection list with some descriptive text for each hook.

Parameters

`handle` Contains a `trc_log_handle_t` data item returned from a successful call to the `trc_open` subroutine.

`hook` Contains a hook id of the form 0xhhh where hhh is the 3-hex-digit hook id. If the `hook` parameter is `TRC_HOOK_ALL`, the names for all hooks in the template file are returned.

`hooknamep` Points to a `trc_hookname_t` structure. The `trc_free` subroutine should be used to free any data referenced by the `trc_hookname_t` data item.

```c
typedef struct {
    trc_hookid_t hookid;
    char *hookname;
} trc_hooknm_t;

typedef struct {
    int trchn_magic;       /* Identifier for this data structure. */
    unsigned trchn_nhooks; /* Number of hooks. */
    trc_hooknm_t *trchn_names; /* Pointer to array of ids and names. */
} trc_hookname_t;
```

Return Values
Upon successful completion, the `trc_hookname` subroutine returns 0.

Error Codes

ENOMEM Not enough memory to satisfy the request.

TRCE_WARN A formatting error was found in the template file. If TRCE_WARN is returned, the function completed.

TRCE_BADFORMAT A formatting error was found in the template file. If TRCE_BADFORMAT was returned, the errno global variable is set to EINVAL.

Related Information

"trc_open Subroutine" on page 439, "trc_loginfo Subroutine" on page 437, "trc_free Subroutine" on page 432, "trc_strerror Subroutine" on page 447, and "trc_perror Subroutine" on page 441.

trc_libcntl Subroutine

Purpose
Performs trace API control functions.

Library
libtrace.a
Syntax

```c
#include <sys/libtrace.h>

int trc_libcntl (handle, cmd, datap);
```

Description

The `trc_libcntl` subroutine provides miscellaneous control functions.

Parameters

- `handle`  
  Contains the handle returned from a successful call to the `trc_open` subroutine.

- `cmd`  
  This is the control function to be performed. Supported functions are:

  **TRC_CNTL_ADJLINENO**
  
  This allows a trace report program to adjust the \$LINENO\ value supplied through the trace templates. Normally, a trace reporting program may assume the \$LINENO\ value is calculated based upon the first line of the output, in `trchi_ascii`, being the first line printed for that hook in the report. If this is not the case, such as with the `2line trcrpt` option, the `SLINENO` value must be adjusted.

  For **TRC_CNTL_ADJLINENO**, the `datap` parameter must contain a signed long value which is added to `SLINENO`. If the value is negative, **TRC_CNTL_ADJLINENO** will decrement the value.

  **TRC_CNTL_NAMELIST**
  
  This allows the namelist to be specified. The default is `/unix`. It does not initialize the symbols, however, and the `trc_libcntl` subroutine returns `EINVAL` if the symbols are already initialized. If symbols are in the trace stream, specified by `trace -n`, those symbols are used regardless of the namelist specification.

  **TRC_CNTL_TEXTOFFSET**
  
  This offsets each line of text, in the `trchi_ascii` data area, by the number of character positions specified, plus `(trchi_indent-1) * 8`; if the associated value is 0, each line is only offset by `(trchi_indent-1) * 8`;

  **TRC_CNTL_TEXTOFFSET_SUBSEQUENT**
  
  This works exactly like **TRC_CNTL_TEXTOFFSET**, except it offsets all lines except the first line of text. The first line is still offset by `(trchi_indent-1) * 8`;

  **TRC_CNTL_PAGESIZE**
  
  This specifies the length of a page.

  **TRC_CNTL_TEXTHEADER**
  
  This specifies a header to be output every page, as specified by the **TRC_CNTL_PAGESIZE** command.

- `datap`  
  Specifies the data parameter.

Return Values

Upon successful completion, the `trc_libcntl` subroutine returns 0.

Error Codes

- **EINVAL**  
  The `handle` or `cmd` parameter is invalid. **EINVAL** is also returned if the value specified with **TRC_CNTL_ADJLINENO** would cause the `SLINENO` value to be negative.
Related Information

The trace daemon in AIX 5L Version 5.3 Commands Reference, Volume 5.

The trcrlp, trcstop, and trcupdate commands in AIX 5L Version 5.3 Commands Reference, Volume 5.

trc_loginfo Subroutine

Purpose

Returns information about a trace log object.

Library

libtrace.a

Syntax

```c
#include <sys/libtrace.h>

int trc_loginfo(int log_object_name, trc_log_info_t *infop)
char *log_object_name;
trc_log_info_t *infop;
```

Description

The trc_loginfo subroutine returns information about the named trace log object. If the log_object_name parameter is NULL or an empty string, the trc_loginfo subroutine returns information about the default log object.

Parameters

- log_object_name
- infop

Names the trace log object. This is specified as it is for the trc_open subroutine. Points to an item of type trc_log_info_t where the information will be returned. The trc_log_info_t structure is defined in the /usr/include/sys/libtrace.h file. It contains such fields as the file size, the time the trace was taken, the trace log file magic number, the command used to start the trace, cpus in the machine, number of cpus traced, multicpu trace indicator (-C), and the trace object type as defined in the trcopen subroutine. The trc_free subroutine should be called to free the trc_loginfo_t information, even if the trc_loginfo subroutine returned an error.

The /usr/include/sys/libtrace.h file contains the data definitions for the returned data, *infop. The following table contains the data item name, data type, and description for each item returned:

<table>
<thead>
<tr>
<th>Label</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trci_magic</td>
<td>int</td>
<td>Structure magic number managed by the library.</td>
</tr>
<tr>
<td>Label</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>trci_logmagic</td>
<td>int</td>
<td>The trace log file's magic number, see the /usr/include/sys/trchdr.h file. This identifies the type of log file, and is included mainly for completeness. The pertinent log file information may be gotten from other fields in this structure.</td>
</tr>
<tr>
<td>trci_time</td>
<td>time_t</td>
<td>The time the trace was taken.</td>
</tr>
<tr>
<td>trci_ipaddr</td>
<td>int</td>
<td>The system's IP address.</td>
</tr>
<tr>
<td>trci_uname</td>
<td>struct utsname</td>
<td>Uname information.</td>
</tr>
<tr>
<td>trci_cmd</td>
<td>char *</td>
<td>The command used to start the trace.</td>
</tr>
<tr>
<td>trci_fnames</td>
<td>trci_fname_t*</td>
<td>Log file names array.</td>
</tr>
<tr>
<td>trci_mach_cpus</td>
<td>int</td>
<td>Number of cpus in the machine.</td>
</tr>
<tr>
<td>trci_traced_cpus</td>
<td>int</td>
<td>Number of traced cpus.</td>
</tr>
<tr>
<td>trci_flags</td>
<td>int</td>
<td>Data stream flags.</td>
</tr>
<tr>
<td>trci_obj_type</td>
<td>int</td>
<td>Trace object type.</td>
</tr>
<tr>
<td>trci_hookids</td>
<td>trc_hookset_t</td>
<td>Binary hook ids map showing the hooks traced. This can be examined with the trc_hkisset subroutine.</td>
</tr>
</tbody>
</table>

The **trci_flags** field contains bit flags as follows:

- **TRCIF_MULTICPU**: This trace was taken with the -C trace option, (for example) it is a multi-cpu trace.
- **TRCIF_64BIT**: This is a 64-bit trace, 32-bit if not set.
- **TRCIF_SEPSEG**: Separate segment buffering was used.
- **TRCIF_CONDTRACE**: Conditional trace by hookid, trace -j, -k, -J, or -K.
- **TRCIF_CONDEXCL**: Trace hook exclusion, -k or -K, was used.

### Return Values
Upon successful completion, the **trc_loginfo** subroutine returns a 0, and information about the trace log object is placed into the memory pointed to by the **infop** parameter.

### Error Codes
Upon error, the **trc_loginfo** subroutine returns information identical to that returned by the **trc_open Subroutine** on page 439.

### Related Information
- **trc_open Subroutine** on page 439, **trc_close Subroutine** on page 427, **trc_read Subroutine** on page 442, **trc_find_first, trc_find_next, and trc_compare Subroutine** on page 428, **trc_seek and trc_tell Subroutine** on page 446, **trc_libcntl Subroutine** on page 435, **trc_strerror Subroutine** on page 447, **trc_errno Subroutine** on page 441, **trcstart Subroutine** on page 453, **trcok Subroutine** on page 452, **trcstop Subroutine** on page 454, and **trc_hkemptyset, trc_hkfillset, trc_hkaddset, trc_hkdelset, and trc_hkisset Subroutine** on page 433.

The **trace daemon** in **AIX 5L Version 5.3 Commands Reference, Volume 5**.

The **trcrl, trcreport, and trcupdate** commands in **AIX 5L Version 5.3 Commands Reference, Volume 5**.
trc_logpath Subroutine

Purpose

Library
libtrace.a

Syntax
#include <sys/libtrace.h>

char *trc_logpath(void)

Description
The trc_logpath subroutine returns the default trace logfile path name. This is normally /var/adm/ras/trcfile, unless changed with the trcctl command or SMIT. Any process that can access and link to the libtrace.a library can call the trc_logpath subroutine and retrieve the current path to the default trace file. With the addition of the trcctl command to the available administration options, system administrators can now set the default to any path rather than always having /var/adm/ras/trcfile as the hard-coded default. Trace Report trcrpt calls the library routines trc_open and trc_loginfo to access the trace file. Beginning with AIX 5.3, trc_open and trc_loginfo both call trc_logpath to access the default file, if it is required. Calling trc_logpath is transparent to trcrpt and the Trace GUI; however, because trc_logpath is available and exported in libtrace.a, other components and third-party products can use it.

Return Values
The trc_logpath subroutine always returns a path name. The path name should be freed, free(path), by the user when appropriate.

Related Information
The trcctl Command.

trc_open Subroutine

Purpose
Opens a trace log object.

Library
libtrace.a

Syntax
#include <sys/libtrace.h>

int trc_open(char *log_object_name, template_file_name, type, handlep)
char *log_object_name, template_file_name;
type;
trc_log_handle_t *handlep;

Description
The trc_open subroutine opens a trace log object. A log object may only be opened for reading.
Two object types are supported, raw and processed. As their names imply, a raw object consists of the raw trace data as it was traced. A processed object consists of data as processed by a trace formatting template file such as the /etc/trcfmt file.

Parameters

**log_object_name**
Specifies the log object to be opened. If this is NULL or an empty string, the default log object, /var/adm/ras/trcfile, is opened. If it is a dash, the input is read from standard input. In this case, the file must be a sequential trace file such as one produced by the `trcrt -r` command, the `-o` trace option, or the trcdead command.

If the file is the base file for a multi-cpu trace, the trace events are merged by the trcrt command, unless the TRC_NOTEMPLATE option was specified. Also, if the file is a single cpu's trace file, it is treated as a single log file.

**template_file_name**
This names the template file. The template file is used if the TRC_LOGPROC type is specified. If NULL, /etc/trcfmt (the default template file) is used. The template file specification is ignored if the TRC_NOTEMPLATE option is specified.

**type**
Consists of flag bits OR'd together. One open type and one object type flag must be specified.

The following is the open type flag:

**TRC_LOGREAD**
Open for reading

The following are the object type flags:

**TRC_LOGRAW**
Specifies that raw trace data is to be read. This data is defined in Debug and Performance Tracing and in the /etc/trcfmt file

**TRC_LOGPROC**
This processes a raw trace log file, one produced by the trace command, using either the trace templates found in the /etc/trcfmt file, or the template file specified by the template_file_name parameter on the trc_open command.

The following are the modifier type flags:

**TRC_LOGVERBATIM**
Returns the file data verbatim, exactly as traced. This is how trcrt -r returns data. See also the TRC_NOTEMPLATE modifier.

**TRC_LIBDEBUG**
Turns on debug mode. This is for IBM® customer support use only.

**TRC_LOGLIVE**
The data returned in the trc_read_t structure is not a unique copy, it is live data. Such data may only be used until the next retrieval API operation. It is not necessary to call the trc_free subroutine to free such data. The TRC_LOGLIVE modifier is used to improve performance when the data read does not need to be retained.

**TRC_RETAIN_HANDLE**
Don't free the handle after an open failure. This allows errors to be processed by the trc_perror or trc_strerror subroutines. The trc_close subroutine must be used to free the file handle.

**TRC_NOTEMPLATE**
Ignore any template file. This is used with the TRC_LOGRAW object flag to prevent any template processing, such as merging multi-cpu trace files. When used in conjunction with the TRC_LOGVERBATIM flag, it causes the retrieval API to return the same data reported with trcrt -r.

**handlep**
Points to the handle returned from a successful call to the trc_open subroutine.
Return Values
Upon successful completion, the `trc_open` subroutine returns a 0 and puts the trace log object handle into the memory pointed to by the `handlep` parameter.

Error Codes
Upon error, the `trc_open` subroutine sets the `errno` global variable to a value in the `errno.h` file, and returns either an `errno.h` value, or an error value defined in the `libtrace.h` file.

- **EINVAL**: Invalid parameter.
- **ENOMEM**: Cannot allocate memory.
- **TRCE_BADFORMAT**: The file is not a valid trace file, and `errno` is set to `EINVAL` if `TRCE_TMPLTFORMAT` is returned. If `TRCE_WARN` is returned, the open succeeded.
- **TRCE_WARN**: The template file contains errors. The `errno` global variable is set to `EINVAL` if `TRCE_TMPLTFORMAT` is returned. If `TRCE_WARN` is returned, the open succeeded.
- **TRCE_TOOMANY**: An internal limit is exceeded. The `errno` global variable is set to `ENOMEM` in this case.

Related Information
- “trc_close Subroutine” on page 427
- “trc_read Subroutine” on page 442
- “trc_loginfo Subroutine” on page 437
- “trc_find_first, trc_find_next, and trc_compare Subroutine” on page 428
- “trc_seek and trc_tell Subroutine” on page 446
- “trc_libcntl Subroutine” on page 435
- “trc_strerror Subroutine” on page 447
- “trc_perror Subroutine,” “trcstart Subroutine” on page 453
- “trcstop Subroutine” on page 454

The trace daemon in AIX 5L Version 5.3 Commands Reference, Volume 5.

The trcpt, trcstop, and trcupdate commands in AIX 5L Version 5.3 Commands Reference, Volume 5.

---

**trc_perror Subroutine**

**Purpose**
Prints all errors associated with a trace log object.

**Library**
libtrace.a

**Syntax**
```c
#include <sys/libtrace.h>

#include <sys/libtrace.h>

void trc_perror (handle, rv, str)
void *handle;
int rv;
char *str;
```

**Description**
The `trc_perror` subroutine works like the `perror` subroutine. If the error in the `rv` parameter is an error from the `errno.h` file, it behaves exactly like the `perror` subroutine.
If there are multiple errors associated with the handle, the trc_perror subroutine prints all errors associated with the object. If the str parameter is NULL, the error’s text is the only text printed. Errors are printed to standard error.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>Contains the handle returned from the call to the trc_open subroutine, the trc_logpos_t object returned by the call to the trc_loginfo subroutine, or NULL. If a handle returned by the trc_open subroutine is passed, the trc_open subroutine need not have been successful, and the TRC_RETAIN_HANDLE option must have been used.</td>
</tr>
<tr>
<td>rv</td>
<td>The return value from a libtrace subroutine.</td>
</tr>
<tr>
<td>str</td>
<td>Used the same as the string passed to the perror subroutine. Errors printed by the trc_perror subroutine are printed as str: error-message.</td>
</tr>
</tbody>
</table>

Related Information

- “trc_open Subroutine” on page 439
- “trc_read Subroutine”
- “trc_loginfo Subroutine” on page 437
- “trc_find_first, trc_find_next, and trc_compare Subroutine” on page 428
- “trc_seek and trc_tell Subroutine” on page 446
- “trc_strerror Subroutine” on page 447
- “trc_hookname Subroutine” on page 434


trc_read Subroutine

**Purpose**
Reads from a trace log object.

**Library**
libtrace.a

**Syntax**

```c
#include <sys/libtrace.h>

int trc_read (handle, ret)
trc_log_handle_t handle;
trc_read_t *ret;
```

**Description**
The trc_read subroutine reads the next sequential data item from the trace log object whose handle is contained in the handle parameter. If the trc_read subroutine follows a trc_find_first or trc_find_next call, it reads the next sequential data item after the one found. To read the next item matching that criteria, use the trc_find_next subroutine.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>Contains the handle returned from a successful call to the trc_open subroutine.</td>
</tr>
</tbody>
</table>
Points to the `trc_read_t` structure to contain the returned information. The raw data will be formatted the same way it is formatted today in the `trcrt` internal data buffer. This is described in the `/etc/trcfmt` file for both 32 and 64 bit events. Thus 32-bit trace items will be formatted as 32-bit items regardless of whether they came from a 32 or 64 bit trace. If `TRC_LOGVERBATIM` was specified, data is returned exactly as traced.

Processed data is the result of trace template processing, see the `/etc/trcfmt` file.

The `trc_free` subroutine should be used to free data referenced from the `trc_read_t` data type. The `trc_free` subroutine need not be used if the `TRC_LOGLIVE` flag was specified when the object was opened.

The `/usr/include/sys/libtrace.h` file contains the data definitions for the returned data.

<table>
<thead>
<tr>
<th>Label</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trcr_magic</td>
<td>int</td>
<td>Trace read data magic number. This is maintained by the library to identify the library version in use.</td>
</tr>
<tr>
<td>trcr_flags</td>
<td>int</td>
<td>Flags that describe the data returned.</td>
</tr>
</tbody>
</table>

The following are definitions for raw data items:

<table>
<thead>
<tr>
<th>Label</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trcri_hookid</td>
<td>trc_hookid_t</td>
<td>Trace hook id of the form 0x0hhh, where hhh is the hook id value, (for example) 134.</td>
</tr>
<tr>
<td>trcri_subhookid</td>
<td>trc_subhookid_t</td>
<td>Subhook id.</td>
</tr>
<tr>
<td>trcri_cpuid</td>
<td>unsigned</td>
<td>The cpu id if known. If the <code>TRCRLF_CPUIDOK</code> flag is set, the cpuid value could be determined, otherwise it should be ignored.</td>
</tr>
<tr>
<td>trcri_tid</td>
<td>unsigned long long</td>
<td>Thread id.</td>
</tr>
<tr>
<td>trcri_timestamp</td>
<td>unsigned long long</td>
<td>Specifies the timestamp in ticks. Use the <code>trc_ticks2nanos</code> function to convert this value to nanoseconds.</td>
</tr>
<tr>
<td>trcri_rawofst</td>
<td>unsigned long long</td>
<td>The offset to the start of this trace item in the trace log file.</td>
</tr>
<tr>
<td>trcri_rawlen</td>
<td>int</td>
<td>The length of the raw data as traced. This is not necessarily the amount of space used for the data in the log file.</td>
</tr>
<tr>
<td>trcri_rawbuf</td>
<td>char *</td>
<td>Pointer to the raw data.</td>
</tr>
</tbody>
</table>

`TRC_LONGD1(r)` - `TRC_LONGD5(r)` return the 5 data words traced by non-generic trace hooks. The `r` value is of type `trc_read_t *`, and must point to a `trc_read_t` item. These macros return unsigned, 64-bit values.

Note: These macros do not check to ensure that the specified register was traced.
The following are definitions for processed data items:

<table>
<thead>
<tr>
<th>Label</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trchi_hookid</td>
<td>trc_hookid_t</td>
<td>The trace hook id of the form 0x0hhh, where hhh is the hook id value, (for example) 134.</td>
</tr>
<tr>
<td>trchi_subhookid</td>
<td>trc_subhookid_t</td>
<td>Subhook id.</td>
</tr>
<tr>
<td>trchi_elapsed_nseconds</td>
<td>unsigned long long</td>
<td>The elapsed time from the start of the trace in nanoseconds.</td>
</tr>
<tr>
<td>trchi_tid</td>
<td>unsigned long long</td>
<td>Thread id.</td>
</tr>
<tr>
<td>trchi_pid</td>
<td>unsigned long long</td>
<td>Process id.</td>
</tr>
<tr>
<td>trchi_svc</td>
<td>unsigned long long</td>
<td>System call address.</td>
</tr>
<tr>
<td>trchi_rawofst</td>
<td>unsigned long long</td>
<td>Offset of the trace event in the log file.</td>
</tr>
<tr>
<td>trchi_trcontime</td>
<td>time64_t</td>
<td>The time of the last TRCON, or this TRCON.</td>
</tr>
<tr>
<td>trchi_trcofftime</td>
<td>time64_t</td>
<td>The time of the last TRCOFF, or this TRCOFF.</td>
</tr>
<tr>
<td>trchi_cpuid</td>
<td>int</td>
<td>cpu id.</td>
</tr>
<tr>
<td>trchi_rcpu</td>
<td>int</td>
<td>cpus remaining in this trace.</td>
</tr>
<tr>
<td>trchi_pri</td>
<td>int</td>
<td>Process priority.</td>
</tr>
<tr>
<td>trchi_intr_depth</td>
<td>int</td>
<td>Interrupt depth.</td>
</tr>
<tr>
<td>trchi_indent</td>
<td>int</td>
<td>The indentation level used by trcrpt. The values are: -1 - SNOPRINT, 0 - no indentation, 1 - application level, 2 - SVC level, 3 - kernel level. Items greater than zero specify the number of tabs, minus 1, that precede each line of the ascii data, see the trchi_ascii field. Each tab represents 8 blanks, so trchi_indent = 2 implies 2 - 1, or 1 tab before each line of data, or 8 blanks.</td>
</tr>
<tr>
<td>trchi_svcname</td>
<td>char *</td>
<td>Current svc name.</td>
</tr>
<tr>
<td>trchi_procname</td>
<td>char *</td>
<td>Current process name.</td>
</tr>
<tr>
<td>trchi_filename</td>
<td>char *</td>
<td>Current file name.</td>
</tr>
<tr>
<td>trchi_ascii</td>
<td>char *</td>
<td>This is the data produced by the trace template for this hook. Each line of data is indented with blanks, according to the trchi_indent value, and the text offset and the subsequent line offset, see the trc_libcntl subroutine.</td>
</tr>
</tbody>
</table>

The trcr_flags field contains bit flags describing characteristics of the returned data. The values are:

<table>
<thead>
<tr>
<th>TRCRF_RAW</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Raw data was read, (for example) the log object was opened with the TRC_LOGRAW open type. Use the raw data items in the return data, (for example) those beginning with trcri_.</td>
</tr>
</tbody>
</table>
TRCRF_PROC: Processed data was read. (for example) the log object was opened with the TRC_LOGPROC open type. Use the processed data items in the return data, (for example) those beginning with trch_.

TRCRF_64BIT: The data is from a 64-bit environment. Note that the trace itself may be from a 32 or 64 bit kernel.

TRCRF_TIMESTAMPED: The entry was timestamped when traced.

TRCRF_CPUIDOK: The cpu id is known. This is always set for a processed entry, and set for a raw entry if the cpuid was contained in each trace hook (see the -p trace command option), or the trace is a multi-cpu trace (see the -C trace option). For a processed trace, the cpu id may not be accurate if the appropriate hooks, 106 and 10C, weren't traced.

TRCRF_GENERIC: This is a generic trace entry, one traced with the TRCGEN or TRCGENT macros. This is set for a raw trace only.

TRCRF_64BITTRACE: This is a 64-bit trace, (for example) it was taken with a 64-bit kernel.

TRCRF_LIVEDATA: The data is live, don't free it. The data will be changed when another read operation is done.

TRCRF_NOPRINT: The associated trace template specified $NOPRINT or $SKIP, (for example) no data should be printed.

**Return Values**

Upon successful completion, the `trc_read` subroutine returns a 0 and puts the data into the `ret` area.

**Error Codes**

Upon error, the `trc_read` subroutine sets the `errno` global variable to a value from `errno.h`, and returns either a value from the `errno.h` file or an error defined in the `libtrace.h` file.

- **EINVAL**: The handle is not valid.
- **TRCE_BADFORMAT**: The trace data is improperly formatted, and the `errno` global variable is set to `EINVAL`.

**Related Information**

- [trc_open Subroutine](#)
- [trc_close Subroutine](#)
- [trc_loginfo Subroutine](#)
- [trc_find_first, trc_find_next, and trc_compare Subroutine](#)
- [trc_libcntl Subroutine](#)
- [trc_strerror Subroutine](#)
- [trc_perror Subroutine](#)
- [trcstart Subroutine](#)
- [trcon Subroutine](#)
- [trcoff Subroutine](#)
- [trcstop Subroutine](#)

The `trace daemon` in *AIX 5L Version 5.3 Commands Reference, Volume 5*.

The `trcrt` `trcstop` and `trcupdate` commands in *AIX 5L Version 5.3 Commands Reference, Volume 5*.

**trc_reg Subroutine**

**Purpose**

Returns register values.
Library
libtrace.a

Syntax
#include <sys/libtrace.h>

int trc_reg(handle, regid, ret)
trc_log_handle_t handle;
int regid;
uint64_t *ret;

Description
The trc_reg subroutine is used to retrieve machine-programmable register values from either a processed
or raw trace entry. It returns a -1 if the specified item was not traced.

trc_reg is only valid for a 64-bit kernel trace.

Parameters
handle Contains the handle returned from a successful trc_open.
regid One of the following reserved register identifiers found in libtrace.h:
   TRC_PURR_ID The PURR register.
   TRC_MCR0_ID, TRC_MCR1_ID, TRC_MCRA_ID The MCR registers, 0, 1, and A.
   TRC_PMCn_ID PMC register n, where n is a value from 1 to 8
ret Points to an unsigned 64-bit integer to hold the return data. If the PURR is
   returned, it is returned in the same units as the elapsed time (that is, ticks for a
   raw trace and nanoseconds for a processed trace).

Return Values
The trc_reg subroutine returns 0 on success; otherwise, it returns the errno value.

Error Codes
EINVAL The specified register ID is invalid.
TRCE_EOF The specified register ID is valid but was not traced.
   Note: TRCE_EOF is the libtrace error for EOF or not found.

Related Information
The trace daemon and trcrpt command.

trc_seek and trc_tell Subroutine

Purpose
Seeks into a trace object and returns the current position that will be used with a future seek.

Library
libtrace.a
Syntax
#include <sys/libtrace.h>

```c
int trc_seek (handle, log_positionp, r);
trc_loghandle_t handle;
trc_logpos_t log_positionp;
trc_read_t *r;

int trc_tell (handle, log_positionp);
trc_loghandle_t handle;
trc_logpos_t *log_positionp;
```

Description
The `trc_seek` subroutine seeks into the log object identified by the `handle` parameter. The `log_positionp` parameter must have been obtained from a previous call to the `trc_tell` subroutine. If the `trc_read_t` pointer, `r`, is not NULL, the `trc_seek` subroutine returns the trace data at the seek point.

The `trc_tell` subroutine creates a `trc_logpos_t` object using the current log position and state.

The `trc_free` subroutine should be used to free a `trc_logpos_t` object that’s no longer needed. However, `trc_free` is not necessary if the `trc_logpos_t` object is passed to another `trc_tell`.

Parameters
- `handle` Contains the handle returned from a successful call to the `trc_open` subroutine.
- `log_positionp` A `trc_logpos_t` returned by a previous call to the `trc_tell` subroutine.
- `r` If not NULL, points to a `trc_read_t` data item where the data at the new position is returned.

Return Values
Upon successful return, the `trc_seek` and `trc_tell` subroutines return 0.

Error Codes
If unsuccessful, the `trc_seek` subroutine returns an i/o error, or `EINVAL` if either the `handle` or `log_positionp` parameter is in error.

Upon error, the `trc_tell` subroutine returns `EINVAL` if the handle is invalid, or `ENOMEM` if storage can’t be obtained for the `trc_logpos_t` object.

Related Information
- “trc_open Subroutine” on page 439
- “trc_read Subroutine” on page 442
- “trc_loginfo Subroutine” on page 437
- “trc_find_first, trc_find_next, and trc_compare Subroutine” on page 428
- “trc_libcntl Subroutine” on page 435
- “trc_strerror Subroutine,” “trc_perror Subroutine” on page 441
- “trc_hookname Subroutine” on page 434


**trc_strerror Subroutine**

**Purpose**
Returns the error message, or next error message, associated with a trace log object or `trc_loginfo` object.
Library
libtrace.a

Syntax
#include <sys/libtrace.h>

char *trc_strerror(handle, rv);
void *handle;
int rv;

Description
The trc_strerror subroutine is similar to the strerror subroutine. If the error in the rv parameter is an error from the errno.h file, it simply returns the string from the strerror subroutine. If the rv parameter is a libtrace error such as TRCE_EOF, it returns the string associated with this error. It is possible for multiple libtrace errors to be present. The trc_strerror subroutine returns the next error in this case. When no more errors are present, the trc_strerror subroutine returns NULL.

Like the strerror subroutine, the trc_strerror subroutine must not be used in a threaded environment.

Parameters
handle Contains the handle returned from the trc_open subroutine, the pointer to a trc_loginfo_t object, or NULL. If a handle returned by the trc_open subroutine is passed, the trc_open subroutine need not have been successful, but the TRC_RETAIN_HANDLE open option must have been used.
rv Contains the return value from a call to the libtrace subroutine.

Return Values
The trc_strerror subroutine returns a pointer to the associated error message. It returns NULL if no more errors are present.

Examples
1. To retrieve all error messages from a call to the trc_open subroutine, call the trc_strerror subroutine as follows:

   
   
   trc_loghandle_t h;
   int rv;
   char *fn, *tfn, *s;

   ...

   rv = trc_open(fn, tfn, TRC_LOGREAD|TRC_LOGPROC|TRC_RETAIN_HANDLE, &h);
   while (rv && s=trc_strerror(h, rv)) {
     fprintf(stderr, "%s\n", s);
   }

2. To accomplish the same thing as the previous example with a single call, do the following:

   
   
   trc_loghandle_t h;
   int rv;
   char *fn, *tfn;

   ...

   rv = trc_open(fn, tfn, TRC_LOGREAD|TRC_LOGPROC|TRC_RETAIN_HANDLE, &h);
   if (rv) trc_perror(h, rv, "");
}
Related Information


trcgen or trcgent Subroutine

Purpose
Records a trace event for a generic trace channel.

Library
Runtime Services Library (librts.a)

Syntax

```c
#include <sys/trchkid.h>

void trcgen(Channel, HkWord, DataWord, Length, Buffer)
unsigned int Channel, HkWord, DataWord, Length;
char *Buffer;

void trcgent(Channel, HkWord, DataWord, Length, Buffer)
unsigned int Channel, HkWord, DataWord, Length;
char *Buffer;
```

Description
The `trcgen` subroutine records a trace event for a generic trace entry consisting of a hook word, a data word, and a variable number of bytes of trace data. The `trcgent` subroutine records a trace event for a generic trace entry consisting of a hook word, a data word, a variable number of bytes of trace data, and a time stamp.

The `trcgen` subroutine and `trcgent` subroutine are located in pinned kernel memory.

Parameters

- **Buffer**
  Specifies a pointer to a buffer of trace data. The maximum size of the trace data is 4096 bytes.
- **Channel**
  Specifies a channel number for the trace session, obtained from the `trcstart` subroutine.
- **DataWord**
  Specifies a word of user-defined data.
**HkWord** Specifies an integer consisting of two bytes of user-defined data (*HkData*), a hook ID (*HkID*), and a hook type (*Hk_Type*).

**HkData** Specifies two bytes of user-defined data.

**HkID** Specifies a hook identifier. For user programs, the hook ID value ranges from 010 to 0FF.

**Hk_Type** Specifies a 4-bit value that identifies the amount of trace data to be recorded:

<table>
<thead>
<tr>
<th>Value</th>
<th>Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hook word</td>
</tr>
<tr>
<td>9</td>
<td>Hook word and a time stamp</td>
</tr>
<tr>
<td>2</td>
<td>Hook word and one data word</td>
</tr>
<tr>
<td>A</td>
<td>Hook word, one data word, and a time stamp</td>
</tr>
<tr>
<td>6</td>
<td>Hook word and up to five data words</td>
</tr>
<tr>
<td>E</td>
<td>Hook word, up to five data words, and a time stamp</td>
</tr>
</tbody>
</table>

**Length** Specifies the length in bytes of the *Buffer* parameter.

**Related Information**

The **trchook** (*trchook, utrchook, trchook64, and utrhook64 Subroutine*) subroutine, **trcoff** (*trcoff Subroutine* on page 452) subroutine, **trcon** (*trcon Subroutine* on page 453) subroutine, **trcstart** (*trcstart Subroutine* on page 453) subroutine, **trcstop** (*trcstop Subroutine* on page 454) subroutine.

The **trace** daemon.

The **trcgenk** kernel service, **trcgenkt** kernel service.

---

**trchook, utrchook, trchook64, and utrhook64 Subroutine**

**Purpose**
Records a trace event.

**Library**
Runtime Services Library (*librts.a*)

**Syntax**

```c
#include <sys/trchkid.h>

void trchook(HkWord, d1, d2, d3, d4, d5);
unsigned int HkWord, d1, d2, d3, d4, d5;

void utrchook(HkWord, d1, d2, d3, d4, d5);
unsigned int HkWord, d1, d2, d3, d4, d5;

void trchook64(HkWord, d1, d2, d3, d4, d5);
unsigned long HkWord, d1, d2, d3, d4, d5;

void utrchook64(HkWord, d1, d2, d3, d4, d5);
unsigned long HkWord, d1, d2, d3, d4, d5;
```
Description

The `trchook` subroutine records a trace event if a trace session is active. Input parameters include a hook word (`HkWord`) and from 0 to 5 words of data. The `trchook` and `trchook64` subroutines are intended for use by the kernel and extensions.

The `utrchook` and `utrchook64` subroutines are intended for programs running at user (application) level.

The `trchook` and `utrchook` subroutines are for use in a 32-bit environment, while the `trchook64` and `utrchook64` subroutines are intended for use in a 64-bit environment. Note that if running a 64-bit application on a 32-bit kernel, the application should use `utrchook64` (the subroutine for its 64-bit environment).

It is strongly recommended that the C macros `TRCHKLn` and `TRCHKLnT` (where `n` is from 0 to 5) be used if possible, instead of calling these subroutines directly.

Parameters

- **d1, d2, d3, d4, d5**
  - Up to 5 words of data from the calling program.
- **HkWord**
  - The `HkWord` parameter has a different format based upon the environment. For the `trchook` and `utrchook` subroutines, it is an unsigned long consisting of a hook ID (`HkID`), a hook type (`Hk_Type`), and two bytes of data from the calling program (`HkData`).
  - **HkID**
    - A hook ID is a 12-bit value. For user programs, the hook ID may be a value from `0x010` to `0x0FF`. Hook identifiers are defined in the `/usr/include/sys/trchkid.h` file.
  - **Hk_Type**
    - A 4-bit value that identifies the amount of trace data to be recorded:
      - **Value**
        - **Records**
          - 1 Hook word
          - 9 Hook word and a time stamp
          - 2 Hook word and one data word
          - A Hook word, one data word, and a time stamp
          - 6 Hook word and up to five data words
          - E Hook word, up to five data words, and a time stamp.
      - **HkData**
        - Two bytes of data from the calling program.

In a 64-bit environment, when using the `trchook64` or `utrchook64` subroutine, the format is `ffffllllhxxssss`, where `f` represents flags, `l` is length, `h` is the hook id, and `s` is the subhook.

The hook and subhook ids are the same as for the 32-bit environment (12-bit hook id and a 16-bit subhook id). Note that the 4 bits between the hook id and subhook are unused.

The flags (the first 16 bits of the 64-bit hookword) are specified as follows:

- **8000** The hook should be timestamped.
- **4000** A generic trace entry, should not use the `trchook64` or `utrchook64` subroutine. For more information see "trcgen or trcgent Subroutine" on page 449.
- **2000** The hook contains 32-bit data. Used by aix trace only.
Automatically include the cpuid when tracing the data.

The length (l) is the second 16 bits of the hookword. It is the length of the data. The length is 0 if no data other than the hookword is traced (TRCHKL0), 8 if one parameter, 8 bytes, is traced (TRCHKL1), 16 for 2 parameters, 24 for 3 parameters, 32 for 4 parameters, and 40 for 5 parameters (TRCHKL5).

**Related Information**

The trcgen subroutine, trcgent subroutine, trcoff subroutine, trcon subroutine, trcstart subroutine, trcstop subroutine.

The trace daemon.

The trcgenk kernel service, trcgenkt kernel service.

---

**trcoff Subroutine**

**Purpose**

Halts the collection of trace data from within a process.

**Library**

Runtime Services Library (librts.a)

**Syntax**

```c
int trcoff(Channel);
int Channel;
```

**Description**

The trcoff subroutine stops trace data collection for a trace channel. The trace session must have already been started using the trace command or the trcstart subroutine.

**Parameters**

Channel Channel number for the trace session.

**Return Values**

If the trcoff subroutine was successful, zero is returned and trace data collection stops. If unsuccessful, a negative one is returned.

**Related Information**

The trcgen subroutine, trchook, utrchook, trchook64, and utrhook64 subroutine, trcstart subroutine, trcstop subroutine.

The trace daemon.

trcgenk kernel service, trcgenkt kernel service.
**trcon Subroutine**

**Purpose**  
Starts the collection of trace data.

**Library**  
Runtime Services Library (librts.a)

**Syntax**  
```c
int trcon( Channel )
int Channel;
```

**Description**  
The `trcon` subroutine starts trace data collection for a trace channel. The trace session must have already been started using the `trace` command or the `trcstart` subroutine.

**Parameters**  
`Channel`  
Specifies one of eight trace channels. Channel number 0 always refers to the Event/Performance trace. Channel numbers 1 through 7 specify generic trace channels.

**Return Values**  
If the `trcon` subroutine was successful, zero is returned and trace data collection starts. If unsuccessful, a negative one is returned.

**Related Information**  
The `trcgenk` kernel service, `trcgenkt` kernel service.

---

**trcstart Subroutine**

**Purpose**  
Starts a trace session.

**Library**  
Runtime Services Library (librts.a)

**Syntax**  
```c
int trcstart( Argument )
char *Argument;
```
Description
The `trcstart` subroutine starts a trace session. The Argument parameter points to a character string containing the flags invoked with the trace daemon. To specify that a generic trace session is to be started, include the -g flag.

Parameters
Argument Character pointer to a string holding valid arguments from the trace daemon.

Return Values
If the trace daemon is started successfully, the channel number is returned. Channel number 0 is returned if a generic trace was not requested. If the trace daemon is not started successfully, a value of -1 is returned.

Files
/dev/trace Trace special file.

Related Information
The trcon subroutine.
The trace daemon.

trcstop Subroutine
Purpose
Stops a trace session.

Library
Runtime Services Library (`lebrts.a`)

Syntax
```c
int trcstop( Channel);
```

Description
The trcstop subroutine stops a trace session for a particular trace channel.

Parameters
Channel Specifies one of eight trace channels. Channel number 0 always refers to the Event/Performance trace. Channel numbers 1 through 7 specify generic trace channels.

Return Values
0 The trace session was stopped successfully.
-1 The trace session did not stop.
Related Information

The `trcgen` subroutine, `trchook`, `trchook64`, and `utrhook64` subroutine, `trcoff` subroutine, `trcon` subroutine, `trcstart` subroutine.

The `trace` daemon.

The `trcgenk` kernel service, `trcgenk` kernel service.

---

`trunc`, `trunck`, or `truncl` Subroutine

**Purpose**

Rounds to truncated integer value.

**Syntax**

```c
#include <math.h>

double trunc (x)
    double x;

float truncf (x)
    float x;

long double truncl (x)
    long double x;
```

**Description**

The `trunc`, `truncf`, and `truncl` subroutines round the `x` parameter to the integer value, in floating format, nearest to but no larger in magnitude than the `x` parameter.

**Parameters**

`x` Specifies the value to be rounded.

**Return Values**

Upon successful completion, the `trunc`, `truncf`, and `truncl` subroutines return the truncated integer value.

If `x` is NaN, a NaN is returned.

If `x` is ±0 or ±Inf, `x` is returned.

**Related Information**

`math.h` in AIX 5L Version 5.3 Files Reference.

---

`truncate`, `truncate64`, `ftruncate`, or `ftruncate64` Subroutine

**Purpose**

Changes the length of regular files or shared memory object.
Library
Standard C Library (libc.a)

Syntax
#include <unistd.h>

int truncate (Path, Length)
const char *Path;
off_t Length;

int ftruncate (FileDescriptor, Length)
int FileDescriptor;
off_t Length;

Note: The truncate64 and ftruncate64 subroutines apply to AIX 4.2 and later releases.

int truncate64 (Path, Length)
const char *Path;
off64_t Length;

int ftruncate64 (FileDescriptor, Length)
int FileDescriptor;
off64_t Length;

Description
Note: The truncate64 and ftruncate64 subroutines apply to AIX 4.2 and later releases.

The truncate and ftruncate subroutines change the length of regular files or shared memory object.

The Path parameter must point to a regular file for which the calling process has write permission. The Length parameter specifies the desired length of the new file in bytes.

The Length parameter measures the specified file in bytes from the beginning of the file. If the new length is less than the previous length, all data between the new length and the previous end of file is removed. If the new length in the specified file is greater than the previous length, data between the old and new lengths is read as zeros. Full blocks are returned to the file system so that they can be used again, and the file size is changed to the value of the Length parameter.

If the file designated in the Path parameter names a symbolic link, the link will be traversed and path-name resolution will continue.

These subroutines do not modify the seek pointer of the file.

These subroutines cannot be applied to a file that a process has open with the O_DEFER flag.

Successful completion of the truncate or ftruncate subroutine updates the st_ctime and st_mtime fields of the file. Successful completion also clears the SetUserID bit (S_ISUID) of the file if any of the following are true:
• The calling process does not have root user authority.
• The effective user ID of the calling process does not match the user ID of the file.
• The file is executable by the group (S_IXGRP) or others (S_IXOTH).

These subroutines also clear the SetGroupID bit (S_ISGID) if:
The file does not match the effective group ID or one of the supplementary group IDs of the process
OR
The file is executable by the owner (S_IXUSR) or others (S_IXOTH).

Note: Clearing of the SetUserID and SetGroupID bits can occur even if the subroutine fails because the data in the file was modified before the error was detected.

truncate and ftruncate can be used to specify any size up to OFF_MAX. truncate64 and ftruncate64 can be used to specify any length up to the maximum file size for the file.

In the large file enabled programming environment, truncate is redefined to be truncate64 and ftruncate is redefined to be ftruncate64.

Parameters

Path Specifies the name of a file that is opened, truncated, and then closed.
FileDescriptor Specifies the descriptor of a file or shared memory object that must be open for writing.
Length Specifies the new length of the truncated file in bytes.

Return Values

Upon successful completion, a value of 0 is returned. If the truncate or ftruncate subroutine is unsuccessful, a value of -1 is returned and the errno global variable is set to indicate the nature of the error.

Error Codes

The truncate and ftruncate subroutines fail if the following is true:

EROFS An attempt was made to truncate a file that resides on a read-only file system.

Note: In addition, the truncate subroutine can return the same errors as the open subroutine if there is a problem opening the file.

Thetruncate and ftruncate subroutines fail if one of the following is true:

EAGAIN The truncation operation fails due to an enforced write lock on a portion of the file being truncated. Because the target file was opened with the O_NONBLOCK or O_NDELAY flags set, the subroutine fails immediately rather than wait for a release.
EDQUOT New disk blocks cannot be allocated for the truncated file. The quota of the user’s or group’s allotted disk blocks has been exhausted on the target file system.
EFBIG An attempt was made to write a file that exceeds the process’ file size limit or the maximum file size. If the user has set the environment variable XPG_SUS_ENV=ON prior to execution of the process, then the SIGXFSZ signal is posted to the process when exceeding the process’ file size limit.
EFBIG The file is a regular file and length is greater than the offset maximum established in the open file description associated with fildes.
EINVAL The file is not a regular file.
EINVAL The Length parameter was less than zero.
EISDIR The named file is a directory.
EINTR A signal was caught during execution.
EIO An I/O error occurred while reading from or writing to the file system.
EMFILE The file is open with O_DEFER by one or more processes.
ENOSPC New disk blocks cannot be allocated for the truncated file. There is no free space on the file system containing the file.
ETXTBSY The file is part of a process that is running.
The named file resides on a read-only file system.

**Notes:**

1. The `truncate` subroutine can also be unsuccessful for other reasons. For a list of additional errors, see "Base Operating System Error Codes For Services That Require Path-Name Resolution".
2. The `truncate` subroutine can return the same errors as the `open` subroutine if there is a problem opening the file.

The `truncate` subroutine fails if the following is true:

- **EBADF** — The `FileDescriptor` parameter is not a valid file descriptor open for writing.
- **EINVAL** — The `FileDescriptor` argument references a file that was opened without write permission.

The `truncate` function will fail if:

- **EACCESS** — 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 of a component of the pathname exceeds `NAME_MAX` bytes.
- **ENOENT** — A component of *path* does not name an existing file or *path* is an empty string.
- **ENTDIR** — A component of the path prefix of *path* is not a directory.
- **EROFS** — The named file resides on a read-only file system.

The `truncate` function may fail if:

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

**Related Information**

The `fclear`, `openx`, `open`, or `creat` subroutine.


Files, Directories, and File Systems for Programmers in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

**tsearch, tdelete, tfind or twalk Subroutine**

**Purpose**

Manages binary search trees.

**Library**

Standard C Library (`libc.a`)

**Syntax**

```c
#include <search.h>
```
void *tsearch (Key, RootPointer, ComparisonPointer)
const void *Key;
void **RootPointer;
int (*ComparisonPointer) (void *Element1, void *Element2);
void *tdelete (Key, RootPointer, ComparisonPointer)
const void *Key;
void **RootPointer;
int (*ComparisonPointer) (void *Element1, void *Element2);
void *tfind (Key, RootPointer, ComparisonPointer)
const void *Key;
void *const *RootPointer;
int (*ComparisonPointer) (void *Element1, void *Element2);
void twalk (Root, Action)
const void *Root;
void (*Action) (const void *Node, VISIT Type, int Level);

Description
The tsearch, tdelete, tfind and twalk subroutines manipulate binary search trees. Comparisons are made with the user-supplied routine specified by the ComparisonPointer parameter. This routine is called with two parameters, the pointers to the elements being compared.

The tsearch subroutine performs a binary tree search, returning a pointer into a tree indicating where the data specified by the Key parameter can be found. If the data specified by the Key parameter is not found, the data is added to the tree in the correct place. If there is not enough space available to create a new node, a null pointer is returned. Only pointers are copied, so the calling routine must store the data. The RootPointer parameter points to a variable that points to the root of the tree. If the RootPointer parameter is the null value, the variable is set to point to the root of a new tree. If the RootPointer parameter is null on entry, then a null pointer is returned.

The tdelete subroutine deletes the data specified by the Key parameter. The RootPointer and ComparisonPointer parameters perform the same function as they do for the tsearch subroutine. The variable pointed to by the RootPointer parameter is changed if the deleted node is the root of the binary tree. The tdelete subroutine returns a pointer to the parent node of the deleted node. If the data is not found, a null pointer is returned. If the RootPointer parameter is null on entry, then a null pointer is returned.

The tfind subroutine searches the binary search tree. Like the tsearch subroutine, the tfind subroutine searches for a node in the tree, returning a pointer to it if found. However, if it is not found, the tfind subroutine will return a null pointer. The parameters for the tfind subroutine are the same as for the tsearch subroutine.

The twalk subroutine steps through the binary search tree whose root is pointed to by the RootPointer parameter. (Any node in a tree can be used as the root to step through the tree below that node.) The Action parameter is the name of a routine to be invoked at each node. The routine specified by the Action parameter is called with three parameters. The first parameter is the address of the node currently being pointed to. The second parameter is a value from an enumeration data type:

typedef enum [preorder, postorder, endorder, leaf] VISIT;

(This data type is defined in the search.h file.) The actual value of the second parameter depends on whether this is the first, second, or third time that the node has been visited during a depth-first, left-to-right traversal of the tree, or whether the node is a leaf. A leaf is a node that is not the parent of another node. The third parameter is the level of the node in the tree, with the root node being level zero.
Although declared as type pointer-to-void, the pointers to the key and the root of the tree should be of type pointer-to-element and cast to type pointer-to-character. Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Points to the data to be located.</td>
</tr>
<tr>
<td>ComparisonPointer</td>
<td>Points to the comparison function, which is called with two parameters that point to the elements being compared.</td>
</tr>
<tr>
<td>RootPointer</td>
<td>Points to a variable that in turn points to the root of the tree.</td>
</tr>
<tr>
<td>Action</td>
<td>Names a routine to be invoked at each node.</td>
</tr>
<tr>
<td>Root</td>
<td>Points to the roots of a binary search node.</td>
</tr>
</tbody>
</table>

**Return Values**

The comparison function compares its parameters and returns a value as follows:

- If the first parameter is less than the second parameter, the *ComparisonPointer* parameter returns a value less than 0.
- If the first parameter is equal to the second parameter, the *ComparisonPointer* parameter returns a value of 0.
- If the first parameter is greater than the second parameter, the *ComparisonPointer* parameter returns a value greater than 0.

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

If the node is found, the *tsearch* and *tfind* subroutines return a pointer to it. If the node is not found, the *tsearch* subroutine returns a pointer to the inserted item and the *tfind* subroutine returns a null pointer. If there is not enough space to create a new node, the *tsearch* subroutine returns a null pointer.

If the *RootPointer* parameter is a null pointer on entry, a null pointer is returned by the *tsearch* and *tdelete* subroutines.

The *tdelete* subroutine returns a pointer to the parent of the deleted node. If the node is not found, a null pointer is returned.

**Related Information**

The *bsearch* subroutine, *hsearch* subroutine, *lsearch* subroutine.

[Searching and Sorting Example Program][1] [Subroutines Overview][2] in *AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs*.

---

**ttylock, ttywait, ttyunlock, or ttylocked Subroutine**

**Purpose**

Controls tty locking functions.

**Library**

Standard C Library (libc.a)
Syntax

```c
int ttylock (DeviceName)
char *DeviceName;
int ttywait (DeviceName)
char *DeviceName;
int ttyunlock (DeviceName)
char *DeviceName;
int ttylocked (DeviceName)
char *DeviceName;
```

Description

The `ttylock` subroutine creates the `LCK..DeviceName` file in the `/etc/locks` directory and writes the process ID of the calling process in that file. If `LCK..DeviceName` exists and the process whose ID is contained in this file is active, the `ttylock` subroutine returns an error.

There are programs like `uucp` and `connect` that create tty locks in the `/etc/locks` directory. The convention followed by these programs is to call the `ttylock` subroutine with an argument of `DeviceName` for locking the `/dev/DeviceName` file. This convention must be followed by all callers of the `ttylock` subroutine to make the locking mechanism work.

The `ttywait` subroutine blocks the calling process until the lock file associated with `DeviceName`, the `/etc/locks/LCK..DeviceName` file, is removed.

The `ttyunlock` subroutine removes the lock file, `/etc/locks/LCK..DeviceName`, if it is held by the current process.

The `ttylocked` subroutine checks to see if the lock file, `/etc/locks/LCK..DeviceName`, exists and the process that created the lock file is still active. If the process is no longer active, the lock file is removed.

Parameters

`DeviceName` Specifies the name of the device.

Return Values

Upon successful completion, the `ttylock` subroutine returns a value of 0. Otherwise, a value of -1 is returned.

The `ttylocked` subroutine returns a value of 0 if no process has a lock on device. Otherwise, a value of -1 is returned.

Examples

1. To create a lock for `/dev/tty0`, use the following statement:
   ```c
   rc = ttylock("tty0");
   ```
2. To lock `/dev/tty0` device and wait for lock to be cleared if it exists, use the following statements:
   ```c
   if (ttylock("tty0"))
      ttywait("tty0");
   rc = ttylock("tty0");
   ```
3. To remove the lock file for device `/dev/tty0` created by a previous call to the `ttylock` subroutine, use the following statement:
   ```c
   ttyunlock("tty0");
   ```
4. To check for a lock on `/dev/tty0`, use the following statement:
   ```c
   rc = ttylocked("tty0");
   ```
Related Information
The `/etc/locks` directory.

The Input and Output Handling Programmer's Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

### ttyname or isatty Subroutine

#### Purpose
Gets the name of a terminal or determines if the device is a terminal.

#### Library
Standard C Library (`libc.a`)  

#### Syntax
```c
#include <unistd.h>

char *ttyname(int FileDescriptor);
int isatty(int FileDescriptor);
```

#### Description
**Attention:** Do not use the `ttyname` subroutine in a multithreaded environment.

The `ttyname` subroutine gets the path name of a terminal.

The `isatty` subroutine determines if the file descriptor specified by the `FileDescriptor` parameter is associated with a terminal.

The `isatty` subroutine does not necessarily indicate that a person is available for interaction, since nonterminal devices may be connected to the communications line.

#### Parameters
- `FileDescriptor` Specifies an open file descriptor.

#### Return Values
The `ttyname` subroutine returns a pointer to a string containing the null-terminated path name of the terminal device associated with the file descriptor specified by the `FileDescriptor` parameter. A null pointer is returned and the `errno` global variable is set to indicate the error if the file descriptor does not describe a terminal device in the `/dev` directory.

The return value of the `ttyname` subroutine may point to static data whose content is overwritten by each call.

If the specified file descriptor is associated with a terminal, the `isatty` subroutine returns a value of 1. If the file descriptor is not associated with a terminal, a value of 0 is returned and the `errno` global variable is set to indicate the error.
Error Codes
The ttyname and isatty subroutines are unsuccessful if one of the following is true:

EBADF    The FileDescriptor parameter does not specify a valid file descriptor.
ENOTTY   The FileDescriptor parameter does not specify a terminal device.

Files
/dev/*    Terminal device special files.

Related Information
The ttyslot subroutine.

The Input and Output Handling Programmer’s Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

ttyslot Subroutine

Purpose
Finds the slot in the utmp file for the current user.

Library
Standard C Library (libc.a)

Syntax
#include <stdlib.h>
int ttyslot (void)

Description
The ttyslot subroutine returns the index of the current user’s entry in the /etc/utmp file. The ttyslot subroutine scans the /etc/utmp file for the name of the terminal associated with the standard input, the standard output, or the error output file descriptors (0, 1, or 2).

The ttyslot subroutine returns -1 if an error is encountered while searching for the terminal name, or if none of the first three file descriptors (0, 1, and 2) is associated with a terminal device.

Files
/etc/inittab    The path to the inittab file, which controls the initialization process.
/etc/utmp     The path to the utmp file, which contains a record of users logged in to the system.

Related Information
The getutent subroutine, ttyname or isatty subroutine.

The Input and Output Handling Programmer’s Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
ulimit Subroutine

Purpose
Sets and gets user limits.

Library
Standard C Library (libc.a)

Syntax
The syntax for the ulimit subroutine when the Command parameter specifies a value of GET_FSIZE or SET_FSIZE is:

```c
#include <ulimit.h>
long int ulimit (Command, NewLimit)
int Command;
off_t NewLimit;
```

The syntax for the ulimit subroutine when the Command parameter specifies a value of GET_DATALIM, SET_DATALIM, GET_STACKLIM, SET_STACKLIM, GET_REALDIR, or SET_REALDIR is:

```c
#include <ulimit.h>
long int ulimit (Command, NewLimit)
int Command;
unsigned long NewLimit;
```

Description
The ulimit subroutine controls process limits.

Even with remote files, the ulimit subroutine values of the process on the client node are used.

Note: Raising the data ulimit does not necessarily raise the program break value. If the proper memory segments are not initialized at program load time, raising your memory limit will not allow access to this memory. Also, without these memory segments initialized, the value returned after such a change may not be the proper break value. If your data limit is RLIM_INFINITY, this value will never advance past the segment size, even if that data is available. Use the -bmaxdata flag of the ld command to set up these segments at load time.
Parameters

Command Specifies the form of control. The following Command parameter values require that the NewLimit parameter be declared as an off_t structure:

**GET_FSIZE (1)**
Returns the process file size limit. The limit is in units of UBSIZE blocks (see the `sys/param.h` file) and is inherited by child processes. Files of any size can be read. The process file size limit is returned in the off_t structure specified by the NewLimit parameter.

**SET_FSIZE (2)**
Sets the process file size limit to the value in the off_t structure specified by the NewLimit parameter. Any process can decrease this limit, but only a process with root user authority can increase the limit. The new file size limit is returned.

The following Command parameter values require that the NewLimit parameter be declared as an integer:

**GET_DATALIM (3)**
Returns the maximum possible break value (as described in the brk or sbrk subroutine).

**SET_DATALIM (1004)**
Sets the maximum possible break value (described in the brk and sbrk subroutines). Returns the new maximum break value, which is the NewLimit parameter rounded up to the nearest page boundary.

**Note:** When a program is executing using the large address-space model, the operating system attempts to modify the soft limit on data size to match the maxdata value. If the maxdata value is larger than the current hard limit on data size, either the program will not execute if the XPG_SUS_ENV environment variable has the value set to ON, or the soft limit will be set to the current hard limit. If the maxdata value is smaller than the size of the program’s static data, the program will not execute.

**GET_STACKLIM (1005)**
Returns the lowest valid stack address.

**Note:** Stacks grow from high addresses to low addresses.

**SET_STACKLIM (1006)**
Sets the lowest valid stack address. Returns the new minimum valid stack address, which is the NewLimit parameter rounded down to the nearest page boundary.

**GET_REALDIR (1007)**
Returns the current value of the real directory read flag. If this flag is a value of 0, a read system call (or readx with Extension parameter value of 0) against a directory returns fixed-format entries compatible with the System V UNIX operating system. Otherwise, a read system call (or readx with Extension parameter value of 0) against a directory returns the underlying physical format.

**SET_REALDIR (1008)**
Sets the value of the real directory read flag. If the NewLimit parameter is a value of 0, this flag is cleared; otherwise, it is set. The old value of the real directory read flag is returned.

**NewLimit** Specifies the new limit. The value and data type or structure of the NewLimit parameter depends on the Command parameter value that is used.

Examples

To increase the size of the stack by 4096 bytes (use 4096 or PAGESIZE), and set the rc to the new lowest valid stack address, enter:

```c
rc = ulimit(SET_STACKLIM, ulimit(GET_STACKLIM, 0) - 4096);
```
Return Values
Upon successful completion, the value of the requested limit is returned. Otherwise, a value of -1 is returned and the errno global variable is set to indicate the error.

All return values are permissible if the ulimit subroutine is successful. To check for error situations, an application should set the errno global variable to 0 before calling the ulimit subroutine. If the ulimit subroutine returns a value of -1, the application should check the errno global variable to verify that it is nonzero.

Error Codes
The ulimit subroutine is unsuccessful and the limit remains unchanged if one of the following is true:

- EPERM A process without root user authority attempts to increase the file size limit.
- EINVAL The Command parameter is a value other than GET_FSIZE, SET_FSIZE, GET_DATALIM, SET_DATALIM, GET_STACKLIM, SET_STACKLIM, GET_REALDIR, or SET_REALDIR.

Related Information
The brk subroutine, sbrk subroutine, getrlimit or setrlimit subroutine, pathconf subroutine, read ("read, readx, ready, readv, or pread Subroutine" on page 29) subroutines, vlimit subroutine, write ("write, writex, writev, writevx or pwrite Subroutines" on page 554) subroutine.

umask Subroutine
Purpose
Sets and gets the value of the file creation mask.

Library
Standard C Library (libc.a)

Syntax
#include <sys/stat.h>

mode_t umask (CreationMask);
mode_t CreationMask;

Description
The umask subroutine sets the file-mode creation mask of the process to the value of the CreationMask parameter and returns the previous value of the mask.

Whenever a file is created (by the open, mkdir, or mknod subroutine), all file permission bits set in the file mode creation mask are cleared in the mode of the created file. This clearing allows users to restrict the default access to their files.

The mask is inherited by child processes.

Parameters

CreationMask Specifies the value of the file mode creation mask. The CreationMask parameter is constructed by logically ORing file permission bits defined in the sys/mode.h file. Nine bits of the CreationMask parameter are significant.
Return Values
If successful, the file permission bits returned by the umask subroutine are the previous value of the file-mode creation mask. The CreationMask parameter can be set to this value in subsequent calls to the umask subroutine, returning the mask to its initial state.

Related Information
The chmod subroutine, mkdir subroutine, mkfifo subroutine, mknod subroutine, openx, open, or creat subroutine, stat ("statx, stat, lstat, fstat, fstatx, fstat, fullstat, fullstat, stat64, lstat64, fstat64, stat64x, fstat64x, or lstat64x Subroutine” on page 318) subroutine.

The sh command, ksh command.

The sys/mode.h file.


Files, Directories, and File Systems for Programmers in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

umount or uvmount Subroutine

Purpose
Removes a virtual file system from the file tree.

Library
Standard C Library (libc.a)

Syntax
int umount (Device)
char *Device;
#include <sys/vmount.h>

int uvmount (VirtualFileSystemID, Flag)
int VirtualFileSystemID;
int Flag;

Description
The umount and uvmount subroutines remove a virtual file system (VFS) from the file tree.

The umount subroutine unmounts only file systems mounted from a block device (a special file identified by its path to the block device).

In addition to local devices, the uvmount subroutine unmounts local or remote directories, identified by the VirtualFileSystemID parameter.

Only a calling process with root user authority or in the system group and having write access to the mount point can umount a device, file and directory mount.
Parameters

- **Device**: The path name of the block device to be unmounted for the `umount` subroutine.
- **VirtualFileSystemID**: The unique identifier of the VFS to be unmounted for the `uvmount` subroutine. This value is returned when a VFS is created by the `vmount` subroutine and may subsequently be obtained by the `mntctl` subroutine. The `VirtualFileSystemID` is also reported in the `stat` subroutine `st_vfs` field.
- **Flag**: Specifies special action for the `uvmount` subroutine. Currently only one value is defined:
  - `UVMNT_FORCE` - Force the unmount. This flag is ignored for device mounts.

Return Values

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned, and the `errno` global variable is set to indicate the error.

Error Codes

The `uvmount` subroutine fails if one of the following is true:

- **EPERM**: The calling process does not have write permission to the root of the VFS, the mounted object is a device or remote, and the calling process does not have root user authority.
- **EINVAL**: No VFS with the specified `VirtualFileSystemID` parameter exists.
- **EBUSY**: A device that is still in use is being unmounted.

The `umount` subroutine fails if one of the following is true:

- **EPERM**: The calling process does not have root user authority.
- **ENOENT**: The `Device` parameter does not exist.
- **ENOBLE**: The `Device` parameter is not a block device.
- **EINVAL**: The `Device` parameter is not mounted.
- **EINVAL**: The `Device` parameter is not local.
- **EBUSY**: A process is holding a reference to a file located on the file system.

The `umount` subroutine can be unsuccessful for other reasons. For a list of additional errors, see "Base Operating System Error Codes For Services That Require Path-Name Resolution".

The `umount` subroutine can be unsuccessful for other reasons. For a list of additional errors, see Appendix A, “Base Operating System Error Codes for Services That Require Path-Name Resolution.”

Related Information

- The `mount` subroutine.
- The `mount` command and `umount` command.


Files, Directories, and File Systems for Programmers in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
uname or unamex Subroutine

Purpose
Gets the name of the current operating system.

Library
Standard C Library (libc.a)

Syntax
#include <sys/utsname.h>
int uname (struct utsname *Name);
int unamex (struct xutsname *Name);

Description
The uname subroutine stores information identifying the current system in the structure pointed to by the Name parameter.

The uname subroutine uses the utsname structure, which is defined in the sys/utsname.h file, and contains the following members:

char sysname[SYS_NMLN];
char nodename[SYS_NMLN];
char release[SYS_NMLN];
char version[SYS_NMLN];
char machine[SYS_NMLN];

The uname subroutine returns a null-terminated character string naming the current system in the sysname character array. The nodename array contains the name that the system is known by on a communications network. The release and version arrays further identify the system. The machine array identifies the system unit hardware being used. The utsname.machine field is not unique if the last two characters in the string are 4C. The character string returned by the uname -Mu command is unique for all systems and the character string returned by the uname -MuL command is unique for all partitions in all systems.

The unamex subroutine uses the xutsname structure, which is defined in the sys/utsname.h file, and contains the following members:

unsigned int nid;
int reserved;
unsigned long long longnid;

The xutsname.nid field is the binary form of the utsname.machine field. The xutsname.nid field is not unique if the last two nibbles are 0x4C. The character string returned by the uname -Mu command is unique for all systems and the character string returned by the uname -MuL command is unique for all partitions in all systems. For local area networks in which a binary node name is appropriate, the xutsname.nid field contains such a name.

Release and version variable numbers returned by the uname and unamex subroutines may change when new BOS software levels are installed. This change affects applications using these values to access licensed programs. Machine variable changes are due to hardware fixes or upgrades.

Contact the appropriate support organization if your application is affected.
Parameters

Name     A pointer to theutsname or xutsname structure.

Return Values
Upon successful completion, the uname or unamex subroutine returns a nonnegative value. Otherwise, a
value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes
The uname and unamex subroutines is unsuccessful if the following is true:
EFAULT     The Name parameter points outside of the process address space.

Related Information
The uname command.

ungetc or ungetwc Subroutine

Purpose
Pushes a character back into the input stream.

Library
Standard C Library (libc.a)

Syntax
#include <stdio.h>

int ungetc (Character, Stream)
int Character;
FILE *Stream;

wint_t ungetwc (Character, Stream)
wint_t Character;
FILE *Stream;

Description
The ungetc and ungetwc subroutines insert the character specified by the Character parameter
(converted to an unsigned character in the case of the ungetc subroutine) into the buffer associated with
the input stream specified by the Stream parameter. This causes the next call to the getc or getwc
subroutine to return the Character value. A successful intervening call (with the stream specified by the
Stream parameter) to a file-positioning subroutine (seek, fsetpos, or rewind) discards any inserted
characters for the stream. The ungetc and ungetwc subroutines return the Character value, and leaves
the file (in its externally stored form) specified by the Stream parameter unchanged.

You can always push one character back onto a stream, provided that something has been read from the
stream or the setbuf subroutine has been called. If the ungetc or ungetwc subroutine is called too many
times on the same stream without an intervening read or file-positioning operation, the operation may not
be successful. The fseek subroutine erases all memory of inserted characters.

The ungetc and ungetwc subroutines return a value of EOF or WEOF if a character cannot be inserted.
A successful call to the `ungetc` or `ungetwc` subroutine clears the end-of-file indicator for the stream specified by the `Stream` parameter. The value of the file-position indicator after all inserted characters are read or discarded is the same as before the characters were inserted. The value of the file-position indicator is decreased after each successful call to the `ungetc` or `ungetwc` subroutine. If its value was 0 before the call, its value is indeterminate after the call.

**Parameters**

- **Character**: Specifies a character.
- **Stream**: Specifies the input stream.

**Return Values**

The `ungetc` and `ungetwc` subroutines return the inserted character if successful; otherwise, `EOF` or `WEOF` is returned, respectively.

**Related Information**

Other wide character I/O subroutines: `fgets` subroutine, `fgetws` subroutine, `fputc` subroutine, `fputwc` subroutine, `fputwc` subroutine, `getwchar` subroutine, `getwc` subroutine, `getws` subroutine, `fputs` subroutine, `fputwc` subroutine, `fputwc` subroutine, `getchar` subroutine, `putwc` subroutine, `putc` subroutine, `putwchar` subroutine.

Related standard I/O subroutines: `fprintf` subroutine, `fread` subroutine, `fopen` subroutine, `fputc` subroutine, `fputs` subroutine, `freopen` subroutine, `fwrite` subroutine, `fgets` subroutine, `fread` subroutine, `freopen` subroutine, `fwrite` subroutine, `fputs` subroutine, `putchar` subroutine, `puts` subroutine, `printf` subroutine, `putc` subroutine, `sprintf` subroutine.

**unlink Subroutine**

**Purpose**

Removes a directory entry.

**Library**

Standard C Library (`libc.a`)

**Syntax**

```c
#include <unistd.h>

int unlink (const char *Path);
```

**Description**

The `unlink` subroutine removes the directory entry specified by the `Path` parameter and decreases the link count of the file referenced by the link. If Network File System (NFS) is installed on your system, this path can cross into another node.
Attention: Removing a link to a directory requires root user authority. Unlinking of directories is strongly discouraged since erroneous directory structures can result. The `rmdir` subroutine should be used to remove empty directories.

When all links to a file are removed and no process has the file open, all resources associated with the file are reclaimed, and the file is no longer accessible. If one or more processes have the file open when the last link is removed, the directory entry disappears. However, the removal of the file contents is postponed until all references to the file are closed.

If the parent directory of `Path` has the `sticky` attribute (described in the `mode.h` file), the calling process must have root user authority or an effective user ID equal to the owner ID of `Path` or the owner ID of the parent directory of `Path`.

The `st_ctime` and `st_mtime` fields of the parent directory are marked for update if the `unlink` subroutine is successful. In addition, if the file's link count is not 0, the `st_ctime` field of the file will be marked for update.

Applications should use the `rmdir` subroutine to remove a directory. If the `Path` parameter names a symbolic link, the link itself is removed.

Parameters

`Path` Specifies the directory entry to be removed.

Return Values

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, the `errno` global variable is set to indicate the error, and the specified file is not changed.

Error Codes

The `unlink` subroutine fails and the named file is not unlinked if one of the following is true:

- **ENOENT** The named file does not exist.
- **EACCES** Write permission is denied on the directory containing the link to be removed.
- **EBUSY** The entry to be unlinked is the mount point for a mounted filesystem, or the file named by `Path` is a named STREAM.
- **EPERM** The file specified by the `Path` parameter is a directory, and the calling process does not have root user authority.
  
  **EPERM** is also returned if the file named by the `Path` parameter is a directory in a JFS2 file system. Note that JFS allows you to unlink a directory.
- **EROFS** The entry to be unlinked is part of a read-only file system.

The `unlink` subroutine can be unsuccessful for other reasons. For a list of additional errors, see Appendix A, "Base Operating System Error Codes for Service That Require Path-Name Resolution".

If NFS is installed on the system, the `unlink` subroutine can also fail if the following is true:

- **ETIMEDOUT** The connection timed out.

Related Information

The `close` subroutine, `link` subroutine, `open` subroutine, `remove` subroutine, `rmdir` subroutine.
unload Subroutine

Purpose
Unloads a module.

Library
Standard C Library (libc.a)

Syntax
#include <sys/ldr.h>

int unload(FunctionPointer)
int (*FunctionPointer)();

Description
The unload subroutine unloads the specified module and its dependents. The value returned by the load subroutine is passed to the unload subroutine as FunctionPointer. The unload subroutine calls termination routines (fini routines) for the specified module and any of its dependents that are not being used by any other module.

The unload subroutine frees the storage used by the specified module only if the module is no longer in use. A module is in use as long as any other module that is in use imports symbols from it.

When a module is unloaded, any deferred resolution symbols that were bound to the module remain bound. These bindings create references to the module that cannot be undone, even with the unload subroutine.

(This paragraph only applies to AIX 4.3.1 and previous releases.) When a process is executing under ptrace control, portions of the process's address space are recopied after the unload processing completes. For a 32-bit process, the main program text (loaded in segment 1) and shared library modules (loaded in segment 13) are recopied. Any breakpoints or other modifications to these segments must be reinserted after the unload call. For a 64-bit process, shared library modules are recopied after an unload call. The debugger will be notified by setting the W_SLWTED flag in the status returned by wait, so that it can reinsert breakpoints.

(This paragraph only applies to AIX 4.3.2 and later releases.) When a process executing under ptrace control calls unload, the debugger is notified by setting the W_SLWTED flag in the status returned by wait. If a module loaded in the shared library is no longer in use by the process, the module is deleted from the process's copy of the shared library segment by freeing the pages containing the module.

Parameters

FunctionPointer Specifies the name of the function returned by the load subroutine.

Return Values
Upon successful completion, the unload subroutine returns a value of 0, even if the module couldn't be unloaded because it is still in use.
Error Codes
If the unload subroutine fails, a value of -1 is returned, the program is not unloaded, and errno is set to indicate the error. errno may be set to one of the following:

EINVAL The FunctionPointer parameter does not correspond to a program loaded by the load subroutine.

Related Information
The load subroutine, loadbind subroutine, loadquery subroutine, dlclose subroutine.

The ld command.

Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

unlockpt Subroutine

Purpose
Unlocks a pseudo-terminal device.

Library
Standard C Library (libc.a)

Syntax
#include <stdlib.h>

int unlockpt (int FileDescriptor);

Description
The unlockpt subroutine unlocks the slave pseudo-terminal device associated with the master pseudo-terminal device defined by the FileDescriptor parameter. This subroutine has no effect if the environment variable XPG_SUS_ENV is not set equal to the string "ON", or if the BSD PTY driver is used.

Parameters
FileDescriptor Specifies the file descriptor of the master pseudo-terminal device.

Return Values
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the errno global variable is set to indicate the error.

Related Information
The grantpt subroutine.

The Input and Output Handling Programmer’s Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
usrinfo Subroutine

Purpose
Gets and sets user information about the owner of the current process.

Library
Standard C Library (libc.a)

Syntax
#include <uinfo.h>

int usrinfo (Command, Buffer, Count);
int Command;
char *Buffer;
int Count;

Description
The usrinfo subroutine gets and sets information about the owner of the current process. The information is a sequence of null-terminated name=value strings. The last string in the sequence is terminated by two successive null characters. A child process inherits the user information of the parent process.

Parameters
Command Specifies one of the following constants:
- GETUINFO
  Copies user information, up to the number of bytes specified by the Count parameter, into the buffer pointed to by the Buffer parameter.
- SETUINFO
  Sets the user information for the process to the number of bytes specified by the Count parameter in the buffer pointed to by the Buffer parameter. The calling process must have root user authority to set the user information.

The minimum user information consists of four strings typically set by the login program:
- NAME=UserName
- LOGIN=LoginName
- LOGNAME=LoginName
- TTY=TTYName

If the process has no terminal, the TTYName parameter should be null.

Buffer Specifies a pointer to a user buffer. This buffer is usually UINFOSIZ bytes long.

Count Specifies the number of bytes of user information copied from or to the user buffer.

Return Values
If successful, the usrinfo subroutine returns a non-negative integer giving the number of bytes transferred. Otherwise, a value of -1 is returned and the errno global variable is set to indicate the error.

Error Codes
The usrinfo subroutine fails if one of the following is true:
- EPERM
  The Command parameter is set to SETUINFO, and the calling process does not have root user authority.
EINVAL The Command parameter is not set to SETINFO or GETINFO.
EINVAL The Command parameter is set to SETINFO, and the Count parameter is larger than UINFOSIZ.
EFAULT The Buffer parameter points outside of the address space of the process.

Related Information
The getuinfo subroutine, setenv subroutine.

The login command.

List of Security and Auditing Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

utimes or utime Subroutine

Purpose
Sets file-access and modification times.

Library
Standard C Library (libc.a)

Syntax
#include <sys/time.h>

int utimes (Path, Times)
char *Path;
struct timeval Times[2];
#include <utime.h>

int utime (Path, Times)
const char *Path;
const struct utimbuf *Times;

Description
The utimes subroutine sets the access and modification times of the file pointed to by the Path parameter to the value of the Times parameter. This subroutine allows time specifications accurate to the second.

The utime subroutine also sets file access and modification times. Each time is contained in a single integer and is accurate only to the nearest second. If successful, the utime subroutine marks the time of the last file-status change (st_ctime) to be updated.

Microsecond time stamps are not implemented, even though the utimes subroutine provides a way to specify them.

Parameters

Path Points to the file.
Specifies the date and time of last access and of last modification. For the `utimes` subroutine, this is an array of `timeval` structures, as defined in the `sys/time.h` file. The first array element represents the date and time of last access, and the second element represents the date and time of last modification. The times in the `timeval` structure are measured in seconds and microseconds since 00:00:00 Greenwich Mean Time (GMT), 1 January 1970, rounded to the nearest second.

For the `utime` subroutine, this parameter is a pointer to a `utimbuf` structure, as defined in the `utime.h` file. The first structure 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 `utimbuf` structure are measured in seconds since 00:00:00 Greenwich Mean Time (GMT), 1 January 1970.

If the `Times` parameter has a null value, the access and modification times of the file are set to the current time. If the file is remote, the current time at the remote node, rather than the local node, is used. To use the call this way, the effective user ID of the process must be the same as the owner of the file or must have root authority, or the process must have write permission to the file.

If the `Times` parameter does not have a null value, the access and modification times are set to the values contained in the designated structure, regardless of whether those times are the same as the current time. Only the owner of the file or a user with root authority can use the call this way.

Return Values
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, the `errno` global variable is set to indicate the error, and the file times are not changed.

Error Codes
The `utimes` or `utime` subroutine fails if one of the following is true:

- **EPERM** The `Times` parameter is not null and the calling process neither owns the file nor has root user authority.
- **EACCES** The `Times` parameter is null, effective user ID is neither the owner of the file nor has root authority, or write access is denied.
- **EROFS** The file system that contains the file is mounted read-only.

The `utimes` or `utime` subroutine can be unsuccessful for other reasons. For a list of additional errors, see [“Base Operating System Error Codes For Services That Require Path-Name Resolution.”](#)

The `utimes` or `utime` subroutine can be unsuccessful for other reasons. For a list of additional errors, see Appendix A, “Base Operating System Error Codes For Services That Require Path-Name Resolution.”

Related Information
The `stat` ("statx, stat, lstat, fstat, fullstat, ffullstat, stat64, lstat64, fstat64, stat64x, fstat64x, or lstat64x Subroutine” on page 318) subroutine.

Files, Directories, and File Systems for Programmers in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

varargs Macros

Purpose
Handles a variable-length parameter list.

Library
Standard C Library (libc.a)
Syntax

```c
#include <stdarg.h>

type va_arg (Argp, Type)
va_list Argp;

void va_start (Argp, ParmN)
va_list Argp;

void va_end (Argp)
va_list Argp;
```

OR

```c
#include <varargs.h>

va_alist Argp;
va_dcl

void va_start (Argp)
va_list Argp;

type va_arg (Argp, Type)
va_list Argp;

void va_end (Argp)
va_list Argp;
```

Description

The varargs set of macros allows you to write portable subroutines that accept a variable number of parameters. Subroutines that have variable-length parameter lists (such as the printf subroutine), but that do not use the varargs macros, are inherently nonportable because different systems use different parameter-passing conventions.

Note: Do not include both <stdarg.h> and <varargs.h>. Use of <varargs.h> is not recommended. It is supplied for backwards compatibility.

For <stdarg.h>

va_start

Initializes the Argp parameter to point to the beginning of the list. The ParmN parameter identifies the rightmost parameter in the function definition. For compatibility with previous programs, it defaults to the address of the first parameter on the parameter list. Acceptable parameters include: integer, double, and pointer. The va_start macro is started before any access to the unnamed arguments.

For <varargs.h>

va_alist

A variable used as the parameter list in the function header.

va_argp

A variable that the varargs macros use to keep track of the current location in the parameter list. Do not modify this variable.

va_dcl

Declaration for va_alist. No semicolon should follow va_dcl.

va_start

Initializes the Argp parameter to point to the beginning of the list.

For <stdarg.h> and <varargs.h>

va_list

Defines the type of the variable used to traverse the list.
va_arg       Returns the next parameter in the list pointed to by the Argp parameter.
va_end       Cleans up at the end.

Your subroutine can traverse, or scan, the parameter list more than once. Start each traversal with a call to the va_start macro and end it with the va_end macro.

Note: The calling routine is responsible for specifying the number of parameters because it is not always possible to determine this from the stack frame. For example, exec is passed a null pointer to signal the end of the list. The printf subroutine determines the number of parameters from its Format parameter.

Parameters

Argp       Specifies a variable that the varargs macros use to keep track of the current location in the parameter list. Do not modify this variable.
Type       Specifies the type to which the expected argument will be converted when passed as an argument. In C, arguments that are char or short should be accessed as int; unsigned char or short arguments are converted to unsigned int, and float arguments are converted to double. Different types can be mixed, but it is up to the routine to know what type of argument is expected, because it cannot be determined at runtime.
ParmN      Specifies a parameter that is the identifier of the rightmost parameter in the function definition.

Examples

The following exec system call implementations are examples of the varargs macros usage.

1. The following example includes <stdarg.h>:

```c
#include <stdarg.h>
define MAXargs 31
int execl (const char *path, ...) {
    va_list Argp;
    char *array[MAXargs];
    int argno=0;
    va_start (Argp, path);
    while ((array[argno++] = va_arg(Argp, char*)) != (char*)0) ;
    va_end(Argp);
    return(execv(path, array));
}
main() {
    execl("/usr/bin/echo", "ArgV[0]", "This", "Is", "A", "Test", "\0"); /* ArgumentV[0] will be discarded by the execv in main(): */ /* by convention ArgV[0] should be a copy of path parameter */
}
```

2. The following example includes <varargs.h>:

```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)
    ; /* Empty loop body */
va_end(ap);
return (execv(file, args));
}

Related Information
The exec subroutines.
The printf subroutine.

List of String Manipulation Services in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

vfscanf, vscanf, or vsscanf Subroutine

Purpose
Formats input of an argument list.

Syntax
#include <stdarg.h>
#include <stdio.h>

int vfscanf (stream, format, arg)
File *restrict stream
const char *format;
va_list arg;

int vscanf (format, arg)
const char *format;
va_list arg;

int vsscanf (format, arg)
const char *format;
va_list arg;

Description
The vscanf, vfscanf, and vsscanf subroutines are equivalent to the scanf, fscanf, and sscanf subroutines, respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined in the <stdarg.h> header file. These subroutines do not invoke the va_end macro. As these functions invoke the va_arg macro, the value of ap after the return is unspecified.

Parameters
stream
format
arg

Return Values
Upon successful completion, these functions shall return the number of successfully matched and assigned input items; this number can be zero in the event of an early matching failure. If the input ends
before the first matching failure or conversion, EOF shall be returned. If a read error occurs, the error indicator for the stream is set, EOF shall be returned, and errno shall be set to indicate the error.

**Related Information**
The “scanf, fscanf, sscanf, or wsscanf Subroutine” on page 126.

### vfscanf, vswscanf, or vwscanf Subroutine

**Purpose**
Wide-character formatted input of the argument list.

**Syntax**

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

int vfscanf (stream, format, arg)
FILE *restrict stream;
const wchar_t *format;
va_list arg;

int vswscanf (ws, format, arg)
const wchar_t *restrict ws;
const wchar_t *format;
va_list arg;

int vwscanf (format, arg)
const wchar_t *format;
va_list arg;
```

**Description**
The `vfscanf`, `vswscanf`, and `vwscanf` subroutines are equivalent to the `fwscanf`, `swscanf`, and `wscanf` subroutines, respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined in the `<stdarg.h>` header file. These subroutines do not invoke the `va_end` macro. As these subroutines invoke the `va_arg` macro, the value of `ap` after the return is unspecified.

**Return Values**
Upon successful completion, the `vfscanf`, `vswscanf`, and `vwscanf` subroutines return the number of successfully matched and assigned input items. This number can be zero 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 the `errno` global variable is set to indicate the error.

**Related Information**

### vwprintf, vwprintf Subroutine

**Purpose**
Wide-character formatted output of a stdarg argument list.
Library
Standard library (libc.a)

Syntax
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vwprintf(const wchar_t *format, va_list arg);
int vfwprintf(FILE *stream, const wchar_t *format, va_list arg);
int vswprintf(wchar_t *s, size_t n, const wchar_t *format, va_list arg);

Description
The vwprintf, vfwprintf and vswprintf functions are the same as wprintf, fprintf and swprintf 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. However, as these functions do invoke the va_arg macro, the value of ap after the return is indeterminate.

Return Values
Refer to fprintf

Error Codes
Refer to fprintf

Related Information
The fprintf subroutine.

---

vmgetinfo Subroutine

Purpose
Retrieves Virtual Memory Manager (VMM) information.

Library
Standard C Library (libc.a)

Syntax
#include <sys/vminfo.h>

int vmgetinfo(void *out, int command, int arg)

Description
The vmgetinfo subroutine returns the current value of certain VMM parameters.

Parameters
arg Additional parameter which depends on the command parameter.
command Specifies which information should be returned. The command parameter has the following valid values:

VMINFO
The content of the vminfo structure (described in sys/vminfo.h) is returned. The out parameter should point to a vminfo structure and arg should be the size of this structure. The smaller of the arg or sizeof(struct vminfo) parameters will be copied.

VM_PAGE_INFO
The size, in bytes, of the page backing the address specified in the addr field of the vm_page_info structure (described in sys/vminfo.h) is returned. The out parameter should point to a vm_page_info structure with the addr field set to the desired address of which to query the page size. The arg parameter should be the size of the vm_page_info structure.

VM_NEW_HEAP_PSIZE
Sets a new preferred page size for future sbreak allocations for the calling process's private data heap. This page size setting is advisory. The out parameter should be a pointer to a psize_t structure that contains the preferred page size, in bytes, to use to back any future sbreak allocations by the calling process. Presently, only 16M (0x1000000) and 4K (0x1000) are supported. The arg parameter should be that of the sizeof(psize_t).

VM_STAGGER_DATA
Staggers the calling process’s current sbreak value by a cumulative per-MCM stagger value. This stagger value must be set through the vmo option data_stagger_interval. The out and arg arguments should be NULL and 0, respectively.

IPC_LIMITS
The content of the ipc_limits struct (described in the sys/vminfo.h file) is returned. The out parameter should point to an ipc_limits structure and arg should be the size of this structure. The smaller of the arg or sizeof(struct ipc_limits) parameters will be copied. The ipc_limits struct contains the inter-process communication (IPC) limits for the system.

out Specifies the address where VMM information should be returned.

Return Values
If the vmgetinfo subroutine is successful, a value of 0 is returned. Otherwise, a value of -1 is returned, and the errno global variable is set to indicate the error.

Error Codes
The vmgetinfo does not succeed if the following are true:

EFAULT The copy operation to the buffer was not successful.
EFAULT Attempt at reading the page size pointed to by the out parameter was not successful.
EINVAL When VM_PAGE_INFO is the command, the addr field of the vm_page_info structure is an invalid address.
EINVAL When VM_NEW_HEAP_PSIZE is the command, the arg parameter is not set to the size of psize_t.
EINVAL When VM_STAGGER_DATA is the command, the out parameter is not set to NULL, or the arg parameter is not set to 0.
EINVAL When VM_STAGGER_DATA is the command, the calling process's data could not be staggered because of resource limitations on the process's data size. (Use ulimit data to increase the allowed data for this process. See the "ulimit Subroutine" on page 464.)
ENOMEM When VM_NEW_HEAP_PSIZE is the command, the break value of the process could not be adjusted because of resource limitations. (See the "ulimit Subroutine" on page 464.)
ENOSYS The command parameter is not valid (or not yet implemented).
ENOSYS Not implemented in current version of AIX (or on 32-bit kernel).
ENOTSUP When VM_NEW_HEAP_PSIZE is the command, the calling process is not 64-bit.
When VM_STAGGER_DATA is the command, the calling process is not 64-bit.

When VM_NEW_HEAP_PSIZE is the command, the user does not have permission to use the requested page size.

Related Information
The "ulimit Subroutine" on page 464.

vmount or mount Subroutine

Purpose
Makes a file system available for use.

Library
Standard C Library (libc.a)

Syntax
#include <sys/types.h>
#include <sys/vmount.h>

int vmount (VMount *, Size);
struct vmount *VMount;
int Size;

int mount (Device, Path, Flags);
char *Device;
char *Path;
int Flags;

Description
The vmount subroutine mounts a file system, thereby making the file available for use. The vmount subroutine effectively creates what is known as a virtual file system. After a file system is mounted, references to the path name that is to be mounted over refer to the root directory on the mounted file system.

A directory can only be mounted over a directory, and a file can only be mounted over a file. (The file or directory may be a symbolic link.)

Therefore, the vmount subroutine can provide the following types of mounts:
• A local file over a local or remote file
• A local directory over a local or remote directory
• A remote file over a local or remote file
• A remote directory over a local or remote directory.

A mount to a directory or a file can be issued if the calling process has root user authority or is in the system group and has write access to the mount point.

To mount a block device, remote file, or remote directory, the calling process must also have root user authority.
The **mount** subroutine only allows mounts of a block device over a local directory with the default file system type. The **mount** subroutine searches the `/etc/filesystems` file to find a corresponding stanza for the desired file system.

**Note:** The **mount** subroutine interface is provided only for compatibility with previous releases of the operating system. The use of the **mount** subroutine is strongly discouraged by normal application programs.

If the directory you are trying to mount over has the sticky bit set to on, you must either own that directory or be the root user for the mount to succeed. This restriction applies only to directory-over-directory mounts.

**Parameters**

**Device**
A path name identifying the block device (also called a special file) that contains the physical file system.

**Path**
A path name identifying the directory on which the file system is to be mounted.

**Flags**
Values that define characteristics of the object to be mounted. Currently these values are defined in the `/usr/include/sys/vmount.h` file:

**MNT_READONLY**
Indicates that the object to be mounted is read-only and that write access is not allowed. If this value is not specified, writing is permitted according to individual file accessibility.

**MNT_NOSUID**
Indicates that `setuid` and `setgid` programs referenced through the mount should not be executable. If this value is not specified, `setuid` and `setgid` programs referenced through the mount may be executable.

**MNT_NODEV**
Indicates that opens of device special files referenced through the mount should not succeed. If this value is not specified, opens of device special files referenced through the mount may succeed.

**VMount**
A pointer to a variable-length **vmount** structure. This structure is defined in the `sys/vmount.h` file.

The following fields of the **VMount** parameter must be initialized before the call to the **vmount** subroutine:

**vmt_revision**
The revision code in effect when the program that created this virtual file system was compiled. This is the value **VMT_REVISION**.

**vmt_length**
The total length of the structure with all its data. This must be a multiple of the word size (4 bytes) and correspond with the **Size** parameter.

**vmt_flags**
Contains the general mount characteristics. The following value may be specified:

**MNT_READONLY**
A read-only virtual file system is to be created.

**vmt_gfstype**
The type of the generic file system underlying the **VMT_OBJECT**. Values for this field are defined in the `sys/vmount.h` file and include:

**MNT_JFS**
Indicates the native file system.
MNT_NFS
Indicates a Network File System client.

MNT_CDROM
Indicates a CD-ROM file system.

vmt_data
An array of structures that describe variable length data associated with the vmount structure. The structure consists of the following fields:

vmt_off
The offset of the data from the beginning of the vmount structure.

vmt_size
The size, in bytes, of the data.

The array consists of the following fields:

vmt_data[VMT_OBJECT]
Specifies the name of the device, directory, or file to be mounted.

vmt_data[VMT_STUB]
Specifies the name of the device, directory, or file to be mounted over.

vmt_data[VMT_HOST]
Specifies the short (binary) name of the host that owns the mounted object. This need not be specified if VMT_OBJECT is local (that is, it has the same vmt_gfstype as / (root), the root of all file systems).

vmt_data[VMT_HOSTNAME]
Specifies the long (character) name of the host that owns the mounted object. This need not be specified if VMT_OBJECT is local.

vmt_data[VMT_INFO]
Specifies binary information to be passed to the generic file-system implementation that supports VMT_OBJECT. The interpretation of this field is specific to the gfs_type.

vmt_data[VMT_ARGS]
Specifies a character string representation of VMT_INFO.

On return from the vmount subroutine, the following additional fields of the VMount parameter are initialized:

vmt_fsid
Specifies the two-word file system identifier; the interpretation of this identifier depends on the gfs_type.

vmt_vfsmnumber
Specifies the unique identifier of the virtual file system. Virtual file systems do not survive the IPL; neither does this identifier.

vmt_time
Specifies the time at which the virtual file system was created.

Size
Specifies the size, in bytes, of the supplied data area.

Return Values
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, and the errno global variable is set to indicate the error.
Error Codes

The `mount` and `vmount` subroutines fail and the virtual file system is not created if any of the following is true:

- **EACCES** The calling process does not have write permission on the stub directory (the directory to be mounted over).
- **EBUSY** `VMT_OBJECT` specifies a device that is already mounted or an object that is open for writing, or the kernel's mount table is full.
- **EFAULT** The `VMount` parameter points to a location outside of the allocated address space of the process.
- **EFBIG** The size of the file system is too big.
- **EFORMAT** An internal inconsistency has been detected in the file system.
- **EINVAL** The contents of the `VMount` parameter are unintelligible (for example, the `vmt_gfstype` is unrecognizable, or the file system implementation does not understand the `VMT_INFO` provided).
- **ENOSYS** The file system type requested has not been configured.
- **ENOTBLK** The object to be mounted is not a file, directory, or device.
- **ENOTDIR** The types of `VMT_OBJECT` and `VMT_STUB` are incompatible.
- **EPERM** `VMT_OBJECT` specifies a block device, and the calling process does not have root user authority.
- **EROFS** An attempt has been made to mount a file system for read/write when the file system cannot support writing.

The `mount` and `vmount` subroutines can also fail if additional errors occur.

Related Information

The `mntctl` subroutine, `umount` subroutine.

The `mount` command, `umount` command.

Files, Directories, and File Systems for Programmers in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

### vsnprintf Subroutine

#### Purpose

Print formatted output.

#### Library

Standard library (`libc.a`)

#### Syntax

```c
#include <stdarg.h>
#include <stdio.h>
int vsnprintf(char *s, size_t n, const char *format, va_list ap)
```

#### Description

Refer to `vfprintf`.

---

### vswprintf Subroutine

#### Purpose

Writes formatted wide characters.
Library
Standard C Library (libc.a)

Syntax
#include <wchar.h>
#include <stdarg.h>

int vwsprintf (wcs, Format, arg)
wchar_t * wcs;
const char * Format;
va_list arg;

Description
The vwsprintf subroutine writes formatted wide characters. It is structured like the vsprintf subroutine with a few differences. One difference is that the wcs parameter specifies a wide character array into which the generated output is to be written, rather than a character array. The second difference is that the meaning of the S conversion specifier is always the same in the case where the # flag is specified. If copying takes place between objects that overlap, the behavior is undefined.

Parameters
wcs Specifies the array of wide characters where the output is to be written.
Format Specifies a multibyte character sequence composed of zero or more directives (ordinary multibyte characters and conversion specifiers). The new formats added to handle the wide characters are:
    %C Formats a single wide character.
    %S Formats a wide character string.
arg Specifies the parameters to be printed.

Return Values
The vwsprintf subroutine returns the number of wide characters (not including the terminating wide character null) written into the wide character array and specified by the wcs parameter.

Related Information
The vsprintf subroutine.
The printf command.


wait, waitpid, wait3, or wait364 Subroutine

Purpose
Waits for a child process to stop or terminate.

Library
Standard C Library (libc.a)
Syntax

```c
#include <sys/wait.h>
pid_t wait (StatusLocation)
int *StatusLocation;
pid_t wait ((void *) 0)
#include <sys/wait.h>

pid_t waitpid (ProcessID, StatusLocation, Options)
int *StatusLocation;
pid_t ProcessID;
int Options;
#include <sys/time.h>
#include <sys/resource.h>
#include <sys/wait.h>

pid_t wait3 (StatusLocation, Options, ResourceUsage)
int *StatusLocation;
int Options;
struct rusage *ResourceUsage;

pid_t wait364 (StatusLocation, Options, ResourceUsage)
int *StatusLocation;
int Options;
struct rusage64 *ResourceUsage;
```

Description

The `wait` subroutine suspends the calling thread until the process receives a signal that is not blocked or ignored, or until any one of the calling process’ child processes stops or terminates. The `wait` subroutine returns without waiting if the child process that has not been waited for has already stopped or terminated prior to the call.

Note: The effect of the `wait` subroutine can be modified by the setting of the `SIGCHLD` signal. See the `sigaction` subroutine for details.

The `waitpid` subroutine includes a `ProcessID` parameter that allows the calling thread to gather status from a specific set of child processes, according to the following rules:

- If the `ProcessID` value is equal to a value of -1, status is requested for any child process. In this respect, the `waitpid` subroutine is equivalent to the `wait` subroutine.
- A `ProcessID` value that is greater than 0 specifies the process ID of a single child process for which status is requested.
- If the `ProcessID` parameter is equal to 0, status is requested for any child process whose process group ID is equal to that of the calling thread’s process.
- If the `ProcessID` parameter is less than 0, status is requested for any child process whose process group ID is equal to the absolute value of the `ProcessID` parameter.

The `waitpid`, `wait3`, and `wait364` subroutine variants provide an `Options` parameter that can modify the behavior of the subroutine. Two values are defined, `WNOHANG` and `WUNTRACED`, which can be combined by specifying their bitwise-inclusive OR. The `WNOHANG` option prevents the calling thread from being suspended even if there are child processes to wait for. In this case, a value of 0 is returned indicating there are no child processes that have stopped or terminated. If the `WUNTRACED` option is set, the call should also return information when children of the current process are stopped because they received a `SIGTTIN`, `SIGTTOU`, `SIGSSTP`, or `SIGTSTOP` signal.
The `wait364` subroutine can be called to make 64-bit `rusage` counters explicitly available in a 32-bit environment.

In AIX 5.1 and later, 64-bit quantities are also available to 64-bit applications through the `wait3()` interface in the `ru_utime` and `ru_stime` fields of `struct rusage`.

When a 32-bit process is being debugged with `ptrace`, the status location is set to `W_SLWTED` if the process calls `load`, `unload`, or `loadbind`. When a 64-bit process is being debugged with `ptrace`, the status location is set to `W_SLWTED` if the process calls `load` or `unload`.

If multiprocessing debugging mode is enabled, the status location is set to `W_SEWTD` if a process is stopped during an exec subroutine and to `W_SFWTED` if the process is stopped during a fork subroutine.

If more than one thread is suspended awaiting termination of the same child process, exactly one thread returns the process status at the time of the child process termination.

If the `WCONTINUED` option is set, the call should return information when the children of the current process have been continued from a job control stop but whose status has not yet been reported.

### Parameters

- **StatusLocation**: Points to integer variable that contains (or will contain) the child process termination status, as defined in the `sys/wait.h` file.
- **ProcessID**: Specifies the child process.
- **Options**: Modifies behavior of subroutine.
- **ResourceUsage**: Specifies the location of a structure to be filled in with resource utilization information for terminated children.

### Macros

The value pointed to by StatusLocation when `wait`, `waitpid`, or `wait3` subroutines are returned, can be used as the ReturnedValue parameter for the following macros defined in the `sys/wait.h` file to get more information about the process and its child process.

- **WIFCONTINUED(ReturnedValue)**
  
  ```c
  pid_t ReturnedValue;
  ```
  
  Returns a nonzero value if status returned for a child process that has continued from a job control stop.

- **WIFSTOPPED(ReturnedValue)**
  
  ```c
  int ReturnedValue;
  ```
  
  Returns a nonzero value if status returned for a stopped child.

- **WSTOPSIG(ReturnedValue)**
  
  ```c
  int ReturnedValue;
  ```
  
  Returns the number of the signal that caused the child to stop.

- **WIFEXITED(ReturnedValue)**
  
  ```c
  int ReturnedValue;
  ```
  
  Returns a nonzero value if status returned for normal termination.

- **WEXITSTATUS(ReturnedValue)**
  
  ```c
  int ReturnedValue;
  ```
  
  Returns the low-order 8 bits of the child exit status.
WIFSIGNALED(ReturnedValue)
int ReturnedValue;

Returns a nonzero value if status returned for abnormal termination.

int WTERMSIG(ReturnedValue)
int ReturnedValue;

Returns the number of the signal that caused the child to terminate.

Return Values
If the wait subroutine is unsuccessful, a value of -1 is returned and the errno global variable is set to indicate the error. In addition, the waitpid, wait3, and wait364 subroutines return a value of 0 if there are no stopped or exited child processes, and the WNOHANG option was specified. The wait subroutine returns a 0 if there are no stopped or exited child processes, also.

Error Codes
The wait, waitpid, wait3, and wait364 subroutines are unsuccessful if one of the following is true:

- **ECHILD** The calling thread’s process has no existing unwaited-for child processes.
- **EINVAL** The value of the Options parameter is not valid.
- **EINTR** This subroutine was terminated by receipt of a signal.
- **EFAULT** The StatusLocation or ResourceUsage parameter points to a location outside of the address space of the process.
- **ECHILD** The process or process group ID specified by the ProcessID parameter does not exist or is not a child process of the calling process.
- **EINVAL** The value of the Options parameter is not valid.

Related Information
The exec subroutine, exit, exit, or atexit subroutine, fork subroutine, getrusage subroutine, pause subroutine, ptrace subroutine, sigaction subroutine, sigaction, sigvec, or signal Subroutine on page 203 subroutine.

waitid Subroutine

Purpose
Waits for a child process to change state.

Library
Standard C Library (libc.a)

Syntax
#include <sys/wait.h>

int waitid(idtype, id, infop, options)
The `waitid` subroutine suspends the calling thread until one child of the process containing the calling thread changes state. It records the current state of a child in the structure pointed to by the `infop` parameter. If a child process changed state prior to the call to the `waitid` subroutine, the `waitid` subroutine returns immediately. If more than one thread is suspended in the `wait` or `waitpid` subroutines waiting for termination of the same process, exactly one thread will return the process status at the time of the target process termination.

### Parameters

- **idtype**: Specifies the child process.
- **id**: Specifies the child process.
- **infop**: Specifies the location of a `siginfo_t` structure to be filled in with resource utilization information.
- **options**: Specifies which state changes the `waitid` subroutine 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 the `infop` parameter in a waitable state. This will not affect the state of the process. The process can be waited for again after this call completes.

### Return Values

If **WNOHANG** was specified and there are no children to wait for, 0 is returned. If the `waitid` subroutine 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.

### Error Codes

The `waitid` subroutine will fail if:

- **ECHILD**: The calling process has no existing unwaited-for child processes.
- **EINTR**: The `waitid` subroutine was interrupted by a signal.
- **EINVAL**: An invalid value was specified for the `options`, or `idtype` parameters and the `id` parameter specifies an invalid set of processes.

### Related Information

The "wait, waitpid, wait3, or wait364 Subroutine" on page 488.
The exec subroutine and _exit, exit, or atexit subroutine in AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 1.

**wcscat, wcschr, wcscmp, wcscpy, or wcscspn Subroutine**

**Purpose**

Performs operations on wide-character strings.

**Library**

Standard C Library (libc.a)

**Syntax**

```c
#include <string.h>

wchar_t * wcscat(WcString1, WcString2)
wchar_t * WcString1;
const wchar_t * WcString2;

wchar_t * wcschr(WcString, WideCharacter)
const wchar_t * WcString;
wchar_t * WideCharacter;

int * wcscmp(WcString1, WcString2)
const wchar_t * WcString1, *WcString2;

wchar_t * wcscpy(WcString1, WcString2)
wchar_t * WcString1;
const wchar_t * WcString2;

size_t wcscspn(WcString1, WcString2)
const wchar_t * WcString1, *WcString2;
```

**Description**

The wcscat, wcschr, wcscmp, wcscpy, or wcscspn subroutine operates on null-terminated wchar_t strings. These subroutines expect the string arguments to contain a wchar_t null character marking the end of the string. A copy or concatenation operation does not perform boundary checking.

The wcscat subroutine appends a copy of the wide-character string pointed to by the WcString2 parameter (including the terminating null wide-character code) to the end of the wide-character string pointed to by the WcString1 parameter. The initial wide-character code of the WcString2 parameter overwrites the null wide-character code at the end of the WcString1 parameter. If successful, the wcscat subroutine returns the WcString1 parameter.

The wcschr subroutine returns a pointer to the first occurrence of the WideCharacter parameter in the WcString parameter. The character value may be a wchar_t null character. The wchar_t null character at the end of the string is included in the search. The wcschr subroutine returns a pointer to the wide character code, if found, or returns a null pointer if the wide character is not found.
The **wcscmp** subroutine compares two **wchar_t** strings. It returns an integer greater than 0 if the **WcString1** parameter is greater than the **WcString2** parameter. It returns 0 if the two strings are equivalent. It returns a number less than 0 if the **WcString1** parameter is less than the **WcString2** parameter. The sign of the difference in value between the first pair of wide-character codes that differ in the objects being compared determines the sign of a nonzero return value.

The **wcsncpy** subroutine copies the contents of the **WcString2** parameter (including the ending **wchar_t** null character) into the **WcString1** parameter. If successful, the **wcsncpy** subroutine returns the **WcString1** parameter. If the **wcsncpy** subroutine copies between overlapping objects, the result is undefined.

The **wcscspn** subroutine computes the number of **wchar_t** characters in the initial segment of the string pointed to by the **WcString1** parameter that do not appear in the string pointed to by the **WcString2** parameter. If successful, the **wcscspn** subroutine returns the number of **wchar_t** characters in the segment.

### Parameters

- **WcString1**: Points to a wide-character string.
- **WcString2**: Points to a wide-character string.
- **WideCharacter**: Specifies a wide character for location.

### Return Values

Upon successful completion, the **wcscat** and **wcsncpy** subroutines return a value of ws1. The **wcschr** subroutine returns a pointer to the wide character code. Otherwise, a null pointer is returned.

The **wcscmp** subroutine returns an integer greater than, equal to, or less than 0, if the wide character string pointed to by the **WcString1** parameter is greater than, equal to, or less than the wide character string pointed to by the **WcString2** parameter.

The **wcscspn** subroutine returns the length of the segment.

### Related Information

The **mbscat** subroutine, **mbschr** subroutine, **mbscmp** subroutine, **mbscpy** subroutine, **mbsrchr** subroutine, **wcscat** ("wcscat, wcsncat, or wcsncpy Subroutine" on page 498) subroutine, **wcsncmp** ("wcscat, wcsncmp, or wcsncpy Subroutine" on page 498) subroutine, **wcsncpy** ("wcscat, wcsncmp, or wcsncpy Subroutine" on page 498) subroutine, **wcsrchr** ("wcsrchr Subroutine" on page 500) subroutine.

### Subroutines, Example Programs, and Libraries

- [National Language Support Overview](#)
- [Multibyte and Wide Character String Comparison Subroutines](#)
- [Understanding Wide Character String Copy Subroutines](#)
- [Wide Character String Search Subroutine](#)

### wcscooll Subroutine

### Purpose

Compares wide character strings.

### Library

- Standard C Library (**libc.a**)

---

494  Technical Reference, Volume 2: Base Operating System and Extensions
Syntax
#include <string.h>

int wcscoll (WcString1, WcString2)
const wchar_t *WcString1, *WcString2;

Description
The wcscoll subroutine compares the two wide-character strings pointed to by the WcString1 and WcString2 parameters based on the collation values specified by the LC_COLLATE environment variable of the current locale.

Note: The wcscoll subroutine differs from the wcscmp subroutine in that the wcscoll subroutine compares wide characters based on their collation values, while the wcscmp subroutine compares wide characters based on their ordinal values. The wcscoll subroutine uses more time than the wcscmp subroutine because it obtains the collation values from the current locale.

The wcscoll subroutine may be unsuccessful if the wide character strings specified by the WcString1 or WcString2 parameter contains characters outside the domain of the current collating sequence.

Parameters
WcString1 Points to a wide-character string.
WcString2 Points to a wide-character string.

Return Values
The wcscoll subroutine returns the following values:

< 0 The collation value of the WcString1 parameter is less than that of the WcString2 parameter.
=0 The collation value of the WcString1 parameter is equal to that of the WcString2 parameter.
>0 The collation value of the WcString1 parameter is greater than that of the WcString2 parameter.

The wcscoll subroutine indicates error conditions by setting the errno global variable. However, there is no return value to indicate an error. To check for errors, the errno global variable should be set to 0, then checked upon return from the wcscoll subroutine. If the errno global variable is nonzero, an error occurred.

Error Codes
EINVAL The WcString1 or WcString2 arguments contain wide-character codes outside the domain of the collating sequence.

Related Information
The wcscmp subroutine.

Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

wcsftime Subroutine

Purpose
Converts date and time into a wide character string.

Library
Standard C Library (libc.a)

Syntax
#include <time.h>

size_t wcsftime (WcString, Maxsize, Format, TimPtr)
wchar_t *WcString;
size_t Maxsize;
const wchar_t *Format;
const struct tm *TimPtr;

Description
The wcsftime function is equivalent to the strftime function, except that:
• The argument wcs points to the initial element of an array of wide-characters into which the generated output is to be placed.
• The argument maxsize indicates the maximum number of wide-characters to be placed in the output array.
• The argument format is a wide-character string and the conversion specifications are replaced by corresponding sequences of wide-characters.
• The return value indicates the number of wide-characters placed in the output array.

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

Parameters
WcString Contains the output of the wcsftime subroutine.
Maxsize Specifies the maximum number of bytes (including the wide character null-terminating byte) that may be placed in the WcString parameter.
Format Specifiers are the same as in strftime (“strftime Subroutine” on page 330) function.
TimPtr Contains the data to be converted by the wcsftime subroutine.

Return Values
If successful, and if the number of resulting wide characters (including the wide character null-terminating byte) is no more than the number of bytes specified by the Maxsize parameter, the wcsftime subroutine returns the number of wide characters (not including the wide character null-terminating byte) placed in the WcString parameter. Otherwise, 0 is returned and the contents of the WcString parameter are indeterminate.

Related Information
The mbstowcs subroutine, strfmon (“strfmon Subroutine” on page 328) subroutine, strftime (“strftime Subroutine” on page 330) subroutine, strptime (“strptime Subroutine” on page 343) subroutine.

Subroutines, Example Programs, and Libraries in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
wcsid Subroutine

Purpose
Returns the charsetID of a wide character.

Library
Standard C Library (libc.a)

Syntax
#include <stdlib.h>

int wcsid (const wchar_t *WC);

Description
The wcsid subroutine returns the charsetID of the wchar_t character. No validation of the character is performed. The parameter must point to a value in the character range of the current code set defined in the current locale.

Parameters
WC  Specifies the character to be tested.

Return Values
Successful completion returns an integer value representing the charsetID of the character. This integer can be a number from 0 through n, where n is the maximum character set defined in the CHARSETID field of the charmap. See “Understanding the Character Set Description (charmap) Source File” in AIX 5L Version 5.3 National Language Support Guide and Reference for more information.

Related Information
The csid subroutine, mbstowcs subroutine.


wcslen Subroutine

Purpose
Determines the number of characters in a wide-character string.

Library
Standard C Library (libc.a)
Syntax
#include <wctr.h>

size_t wcslen(const wchar_t *WcString);

Description
The wcslen subroutine computes the number of wchar_t characters in the string pointed to by the WcString parameter.

Parameters
WcString Specifies a wide-character string.

Return Values
The wcslen subroutine returns the number of wchar_t characters that precede the terminating wchar_t null character.

Related Information
The mbslen subroutine, wctomb subroutine.

wcsncat, wcsncmp, or wcsncpy Subroutine

Purpose
Performs operations on a specified number of wide characters from one string to another.

Library
Standard C Library (libc.a)

Syntax
#include <wctr.h>

wchar_t * wcscat(wchar_t * WcString1, const wchar_t * WcString2, size_t Number);
const wchar_t * wcsncpy(wchar_t * WcString1, const wchar_t * WcString2, size_t Number);
size_t wcsncmp(const wchar_t * WcString1, wchar_t * WcString2, size_t Number);
The wcsncat, wcsncmp and wcsncpy subroutines operate on null-terminated wide character strings.

The wcsncat subroutine appends characters from the WcString2 parameter, up to the value of the Number parameter, to the end of the WcString1 parameter. It appends a wchar_t null character to the result and returns the WcString1 value.

The wcsncmp subroutine compares wide characters in the WcString1 parameter, up to the value of the Number parameter, to the WcString2 parameter. It returns an integer greater than 0 if the value of the WcString1 parameter is greater than the value of the WcString2 parameter. It returns a 0 if the strings are equivalent. It returns an integer less than 0 if the value of the WcString1 parameter is less than the value of the WcString2 parameter.

The wcsncpy subroutine copies wide characters from the WcString2 parameter, up to the value of the Number parameter, to the WcString1 parameter. It returns the value of the WcString1 parameter. If the number of characters in the WcString2 parameter is less than the Number parameter, the WcString1 parameter is padded out with wchar_t null characters to a number equal to the value of the Number parameter.

Parameters

- WcString1: Specifies a wide-character string.
- WcString2: Specifies a wide-character string.
- Number: Specifies the range of characters to process.

Related Information

The mbsncat subroutine, mbsncmp subroutine, mbsncpy subroutine, wcscat subroutine, wcschr subroutine, wcscmp subroutine, wcscpy subroutine, or wcscspn subroutine on page 493 subroutine, wcsncpy subroutine on page 493 subroutine.

Subroutines, Example Programs, and Libraries in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.


wcspbrk Subroutine

Purpose

Locates the first occurrence of characters in a string.

Library

Standard C Library (libc.a)
Syntax

```c
#include <string.h>

wchar_t *wcspbrk(WcString1, WcString2)
const wchar_t *WcString1;
const wchar_t *WcString2;
```

Description
The **wcspbrk** subroutine locates the first occurrence in the wide character string pointed to by the **WcString1** parameter of any wide character from the string pointed to by the **WcString2** parameter.

Parameters
- **WcString1**: Points to a wide-character string being searched.
- **WcString2**: Points to a wide-character string.

Return Values
If no **wchar_t** character from the **WcString2** parameter occurs in the **WcString1** parameter, the **wcspbrk** subroutine returns a pointer to the wide character, or a null value.

Related Information
The **mbspbrk** subroutine, **wcschr** subroutine, **wcsspn** subroutine, **wcsrchr** subroutine, **wcsrchr Subroutine**, **wcscssn** subroutine, **wcstok** subroutine, **wcswcs** subroutine, **wcswcs Subroutine**.

Subroutines, Example Programs, and Libraries in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.


---

**wcsrchr Subroutine**

Purpose
Locates a **wchar_t** character in a wide-character string.

Library
Standard C Library (**libc.a**)

Syntax

```c
#include <wctype.h>

wchar_t *wcsrchr ( WcString, WideCharacter)
const wchar_t *WcString;
wint_t WideCharacter;
```
Description
The **wcsrchr** subroutine locates the last occurrence of the *WideCharacter* value in the string pointed to by the *WcString* parameter. The terminating *wchar_t* null character is considered to be part of the string.

Parameters

| **WcString** | Points to a string. |
| **WideCharacter** | Specifies a *wchar_t* character. |

Return Values
The **wcsrchr** subroutine returns a pointer to the *WideCharacter* parameter value, or a null pointer if that value does not occur in the specified string.

Related Information
The **mbsschr** subroutine, **mbsrchr** subroutine, **wcschr** subroutine, **wcscspn** subroutine, **wcsbrk** subroutine, **wcsspn** subroutine, **wcstok** subroutine, **wcssp** subroutine, **wcswcs** subroutine.

Subroutines, Example Programs, and Libraries in **AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs**.


**wcsrtombs** Subroutine

Purpose
Convert a wide-character string to a character string (restartable).

Library
Standard library (**libc.a**)?

Syntax

```c
#include <wchar.h>

size_t wcsrtombs (char * dst, const wchar_t ** src, size_t len, mbstate_t * ps);
```

Description
The **wcsrtombs** function converts a sequence of wide-characters from the array indirectly pointed to by **src** into a sequence of corresponding characters, beginning in the conversion state described by the object pointed to by **ps**. If **dst** is not a null pointer, the converted characters are then stored into the array pointed to by **dst**. Conversion continues up to and including a terminating null wide-character, which is also stored. Conversion stops earlier in the following cases:

- When a code is reached that does not correspond to a valid character.
- When the next character would exceed the limit of **len** total bytes to be stored in the array pointed to by **dst** (and **dst** is not a null pointer).

Each conversion takes place as if by a call to the **wcrtom** function.
If \texttt{dst} is not a null pointer, the pointer object pointed to by \texttt{src} is assigned either a null pointer (if conversion stopped due to reaching a terminating null wide-character) or the address just past the last wide-character converted (if any). If conversion stopped due to reaching a terminating null wide-character, the resulting state described is the initial conversion state.

If \texttt{ps} is a null pointer, the \texttt{wcsrtombs} function uses its own internal \texttt{mbstate_t} object, which is initialised at program startup to the initial conversion state. Otherwise, the \texttt{mbstate_t} object pointed to by \texttt{ps} is used to completely describe the current conversion state of the associated character sequence. The implementation will behave as if no function defined in this specification calls \texttt{wcsrtombs}.

The behavior of this function is affected by the LC\_CTYPE category of the current locale.

**Return Values**

If conversion stops because a code is reached that does not correspond to a valid character, an encoding error occurs. In this case, the \texttt{wcsrtombs} function stores the value of the macro EILSEQ in \texttt{errno} and returns (\texttt{size_t})-1; the conversion state is undefined. Otherwise, it returns the number of bytes in the resulting character sequence, not including the terminating null (if any).

**Error Codes**

The \texttt{wcsrtombs} function may fail if:

- \texttt{EINVAL} \texttt{ps} points to an object that contains an invalid conversion state.
- \texttt{EILSEQ} A wide-character code does not correspond to a valid character.

**Related Information**

The \texttt{wctomb} subroutine.

### wcsppn Subroutine

**Purpose**

Returns the number of wide characters in the initial segment of a string.

**Library**

Standard C Library (\texttt{libc.a})

**Syntax**

```c
#include <wcstr.h>

size_t wcsppn(const wchar_t *WcString1, wchar_t *WcString2);
```

**Description**

The \texttt{wcsppn} subroutine computes the number of \texttt{wchar_t} characters in the initial segment of the string pointed to by the \texttt{WcString1} parameter. The \texttt{WcString1} parameter consists entirely of \texttt{wchar_t} characters from the string pointed to by the \texttt{WcString2} parameter.

**Parameters**

- \texttt{WcString1} Points to the initial segment of a string.
- \texttt{WcString2} Points to a set of characters string.
Return Values
The wcspsn subroutine returns the number of wchar_t characters in the segment.

Related Information
The wcschr subroutine, wcschr, wcscmp, wcscpy, or wcspsn subroutine, wcspsn subroutine, wcschr subroutine, wcspbrk subroutine, wcsrchr subroutine, wcstok subroutine, wcswcs subroutine.

Subroutines, Example Programs, and Libraries in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.


wcsstr Subroutine

Purpose
Find a wide-character substring.

Library
Standard library (libc.a)

Syntax
#include <wchar.h>

wchar_t *wcsstr (const wchar_t *ws1, const wchar_t *ws2);

Description
The wcsstr function locates the first occurrence in the wide-character string pointed to by ws1 of the sequence of wide-characters (excluding the terminating null wide-character) in the wide-character string pointed to by ws2.

Return Values
On successful completion, wcsstr 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.

wcstod, wcstof, or wcstold Subroutine

Purpose
Converts a wide character string to a double-precision number.

Library
Standard C Library (libc.a)

Syntax
#include <stdlib.h>
#include <wchar.h>

wcstod, wcstof, or wcstold subroutine.
double wcstod (nptr, endptr)
const wchar_t *nptr;
wchar_t **endptr;

float wcstof (nptr, endptr)
const wchar_t *restrict nptr;
wchar_t **restrict endptr;

long double wcstold (nptr, endptr)
const wchar_t *restrict format;
wchar_t **restrict nptr;

Description
The wcstod, wcstof, and wcstold subroutines convert the initial portion of the wide-character string pointed to by nptr to double, float and long double representation, respectively. First, they decompose the input wide-character string into three parts:

- An initial, possibly empty, sequence of white-space wide-character codes.
- A subject sequence interpreted as a floating-point constant or representing infinity or NaN.
- A final wide-character string of one or more unrecognized wide-character codes, including the terminating null wide-character code of the input wide-character string.

Then they convert the subject sequence to a floating-point number, and return the result.

The expected form of the subject sequence is an optional plus or minus sign, and one of the following:

- A non-empty sequence of decimal digits optionally containing a radix character, and an optional exponent part.
- A 0x or 0X, and a non-empty sequence of hexadecimal digits optionally containing a radix character, and an optional binary exponent part.
- One of INF or INFINITY, or any other wide string equivalent except for case.
- One of NAN or NAN(n-wchar-sequence opt), or any other wide string ignoring case in the NAN part, where:

  n-wchar-sequence:
  digit
  nondigit
  n-wchar-sequence digit
  n-wchar-sequence nondigit

The subject sequence is defined as the longest initial subsequence of the input wide string, starting with the first non-white-space wide character, that is of the expected form. The subject sequence contains no wide characters if the input wide string is not of the expected form.

If the subject sequence has the expected form for a floating-point number, the sequence of wide characters starting with the first digit or the radix character (whichever occurs first) are interpreted as a floating constant according to the rules of the C language, except that the radix character is used in place of a period. If neither an exponent part or a radix character appears in a decimal floating-point number, or if a binary exponent part does not appear in a hexadecimal floating-point number, an exponent part of the appropriate type with value zero is assumed to follow the last digit in the string.

If the subject sequence begins with a minus sign, the sequence is interpreted as negated. A wide-character sequence INF or INFINITY is interpreted as an infinity, if representable in the return type, or else as if it were a floating constant that is too large for the range of the return type. A wide-character sequence NAN or NAN(n-wchar-sequence opt) is interpreted as a quiet NaN, if supported in the return type, or else as if it were a subject sequence part that does not have the expected form. The meaning of
the \textit{n}-wchar sequences is implementation-defined. A pointer to the final wide string is stored in the object pointed to by \textit{endptr}, provided that \textit{endptr} is not a null pointer.

If the subject sequence has the hexadecimal form and FLT\_RADIX is a power of 2, the conversion will be rounded in an implementation-defined manner.

The radix character is as defined in the program’s locale (category \textit{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 C or POSIX locales, other implementation-defined subject sequences may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed. The value of \textit{nptr} is stored in the object pointed to by \textit{endptr}, provided that \textit{endptr} is not a null pointer.

The \texttt{wcstod}, \texttt{wcstof}, and \texttt{wcstold} subroutines do not change the setting of the \texttt{errno} global variable if successful.

Since 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, call \texttt{wcstod}, \texttt{wcstof}, or \texttt{wcstold}, and check \texttt{errno}.

\textbf{Parameters}

\begin{itemize}
  \item \texttt{nptr} Contains a pointer to the wide character string to be converted to a double-precision value.
  \item \texttt{endptr} Contains a pointer to the position in the string specified by the \textit{nptr} parameter where a wide character is found that is not a valid character for the purpose of this conversion.
\end{itemize}

\textbf{Return Values}

Upon successful completion, the \texttt{wcstod}, \texttt{wcstof}, and \texttt{wcstold} subroutines return the converted value. If no conversion could be performed, 0 is returned and the \texttt{errno} global variable may be set to \texttt{EINVAL}.

If the correct value is outside the range of representable values, plus or minus \texttt{HUGE\_VAL}, \texttt{HUGE\_VALF}, or \texttt{HUGE\_VALL} is returned (according to the sign of the value), and \texttt{errno} is set to \texttt{ERANGE}.

If the correct value would cause underflow, a value whose magnitude is no greater than the smallest normalized positive number in the return type is returned and \texttt{errno} set to \texttt{ERANGE}.

\textbf{Related Information}

\begin{itemize}
  \item \texttt{scanf, fscanf, sscanf, or wscanf Subroutine} on page 126, \texttt{setlocale Subroutine} on page 171, and \texttt{strtol, strtoul, strtoll, strtoffl, or atoi Subroutine} on page 341.
  \item \texttt{ctype, isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, isgraph, iscntrl, or isascii Subroutines} and \texttt{localeconv Subroutine} in \textit{AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 1}.
\end{itemize}

\textbf{wcstoimax or wcstoumax Subroutine}

\textbf{Purpose}

Converts a wide-character string to an integer type.

\textbf{Syntax}

\begin{verbatim}
#include <stddef.h>
#include <inttypes.h>
\end{verbatim}
intmax_t wcstoimax (nptr, endptr, base)
const wchar_t *restrict nptr;
wchar_t **restrict endptr;
int base;

uintmax_t wcstoumax (nptr, endptr, base)
const wchar_t *restrict nptr;
wchar_t **restrict endptr;
int base;

Description
The wcstoimax or wcstoumax subroutines are equivalent to the wcstol, wcstoll, wcstoul, and wcstoull subroutines, respectively, except that the initial portion of the wide string is converted to intmax_t and uintmax_t representation, respectively.

Parameters
nptr Points to the wide-character string.
endptr Points to the object where the final wide-character string is stored.
base Determines the subject sequence interpreted as an integer.

Return Values
The wcstoimax or wcstoumax subroutines return the converted value, if any.

If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, {INTMAX_MAX}, {INTMAX_MIN}, or {UINTMAX_MAX} is returned (according to the return type and sign of the value, if any), and the errno global variable is set to ERANGE.

Related Information
The "wcstol or wcstoll Subroutine" on page 508.

inttypes.h in AIX 5L Version 5.3 Files Reference.

wcstok Subroutine

Purpose
Converts wide-character strings to tokens.

Library
Standard C Library (libc.a)

Syntax
#include <wchar.h>

wchar_t *wcstok (WcString1, WcString2, ptr)
wchar_t *WcString1;
const wchar_t *WcString2;
wchar_t **ptr

Description
A sequence of calls to the wcstok subroutine breaks the wide-character string pointed to by WcString1 into a sequence of tokens, each of which is delimited by a wide-character code from the wide-character
string pointed to by WcString2. The third argument points to a caller-provided wchar_t pointer where wcstok stores information necessary for it to continue scanning the same wide-character string.

The first call in the sequence has WcString1 as its first argument and is followed by calls with a nullpointer as their first argument. The separator string pointed to by WcString2 may be different from call to call.

The first call in the sequence searches the wide-character string pointed to by WcString1 for the first wide-character code that is not contained in the current separator string pointed to by WcString2. If no such wide-character code is found, then there are no tokens in the wide-character string pointed to by WcString1 and wcstok returns a null pointer. If such a wide-character code is found, it is the start of the first token.

The wcstok subroutine 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 WcString1, and subsequent searches for a token returns 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 subroutine saves a pointer to the following wide-character code, from which the next search for a token starts.

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 behaves as if no function calls wcstok.

Parameters

ptr Contains a pointer to a caller-provided wchar_t pointer where wcstok stores information necessary for it to continue scanning the same wide-character string.

WcString1 Contains a pointer to the wide-character string to be searched.

WcString2 Contains a pointer to the string of wide-character token delimiters.

Return Values

Upon successful completion, wcstok returns a pointer to the first wide-character code of a token. Otherwise, if there is no token, wcstok returns a null pointer.

Examples

To convert a wide-character string to tokens, use the following:

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

main()
{
    wchar_t WCString1[] = L"a??b,,,c";
    wchar_t *ptr;
    wchar_t *pwcs;

    (void) setlocale(LC_ALL, "");
    pwcs = wcstok(WCString1, L"?", &ptr);
    /* pwcs points to the token L"a"*/
    pwcs = wcstok((wchar_t *)NULL, L",", &ptr);
    /* pwcs points to the token L"??b"*/
    pwcs = wcstok((wchar_t *)NULL, L"#", &ptr);
    /* pwcs points to the token L"c"*/
} 
```
The \texttt{wcstol} subroutine converts a wide-character string to a long integer representation. The \texttt{wcstoll} subroutine converts a wide-character string to a long long integer representation.

1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by the \texttt{iswspace} subroutine)
2. A subject sequence interpreted as an integer and represented in a radix determined by the \texttt{Base} parameter
3. A final wide-character string of one or more unrecognized wide-character codes, including the terminating wide-character null of the input wide-character string

If possible, the subject is then converted to an integer, and the result is returned.

The \texttt{Base} parameter can take the following values: 0 through 9, or a (or A) through z (or Z). There are potentially 36 values for the base. If the base value is 0, the expected form of the subject string is that of a decimal, octal, or hexadecimal constant, any of which can be preceded by a + (plus sign) or - (minus sign). A decimal constant starts with a non zero digit, and is composed 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. A hexadecimal
constant is defined as the prefix 0x (or 0X) followed by a sequence of decimal digits and the letters a (or A) to f (or F) with values ranging from 10 (for a or A) to 15 (for f or F).

If the base value is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer in the radix specified by the Base parameter, optionally preceded by a + or -, but not including an integer suffix. The letters a (or A) through z (or Z) are ascribed the values of 10 to 35. Only letters whose values are less than that of the base are permitted. If the value of base is 16, the characters 0x or 0X may optionally precede the sequence of letters or digits, following the sign, if present.

The wide-character string is parsed to skip the initial space characters (as determined by the iswspace subroutine). Any non-space character signifies the start of a subject string that may form an integer in the radix specified by the Base parameter. The subject sequence is defined to be the longest initial substring that is a long integer of the expected form. Any character not satisfying this form begins the final portion of the wide-character string pointed to by the Endptr parameter on return from the call to the wcstol or wcstoll subroutine.

Parameters

Nptr Contains a pointer to the wide-character string to be converted to a long integer number.
Endptr Contains a pointer to the position in the Nptr parameter string where a wide-character is found that is not a valid character.
Base Specifies the radix in which the characters are interpreted.

Return Values

The wcstol and wcstoll subroutines return the converted value of the long or long long integer if the expected form is found. If no conversion could be performed, a value of 0 is returned. If the converted value is outside the range of representable values, LONG_MAX or LONG_MIN is returned for the wcstol subroutine and LLONG_MAX or LLONG_MIN is returned for the wcstoll subroutine (according to the sign of the value). The value of errno is set to ERANGE. If the base value specified by the Base parameter is not supported, EINVAL is returned.

If the subject sequence has the expected form, it is interpreted as an integer constant in the appropriate base. A pointer to the final string is stored in the Endptr parameter if that parameter is not a null pointer.

If the subject sequence is empty or does not have a valid form, no conversion is done. The value of the Nptr parameter is stored in the Endptr parameter if that parameter is not a null pointer.

Since 0, LONG_MIN, and LONG_MAX (for wcstol) and LLONG_MIN, and LLONG_MAX (for wcstoll) are returned in the event of an error and are also valid returns if the wcstol or wcstoll subroutine is successful, applications should set the errno global variable to 0 before calling either subroutine, and check errno after return. If the errno global value has changed, an error occurred.

Examples

To convert a wide-character string to a signed long integer, use the following code:

```c
#include <stdlib.h>
#include <locale.h>
#include <errno.h>

main()
{
    wchar_t *WCString, *endptr;
    long int retval;
    (void)setlocale(LC_ALL,""甦;
    /**<Set errno to 0 so a failure for wcstol can be
    *detected*/
```
errno=0;
/*
 * Let WCString point to a wide character null terminated
 * string containing a signed long integer value
 * */
    retval = wcstol ( WCString &endptr, 0 );
/* Check errno, if it is non-zero, wcstol failed */
if (errno != 0) {
    /*Error handling*/
} else if (&WCString == endptr) {
/* No conversion could be performed */
    /* Handle this case accordingly. */
}
/* retval contains long integer */

Related Information

The iswspace subroutine, wcstod "wcstod, wcstof, or wcstold Subroutine" on page 503 subroutine, wcstoul "wcstoul or wcstoull Subroutine" on page 511 subroutine.

Subroutines, Example Programs, and Libraries in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.


wcstombs Subroutine

Purpose

Converts a sequence of wide characters into a sequence of multibyte characters.

Library

Standard C Library (libc.a)

Syntax

#include <stdlib.h>

size_t wcstombs (String, WCString, Number)
char *String;
const wchar_t *WCString;
size_t Number;

Description

The wcstombs subroutine converts the sequence of wide characters pointed to by the WCString parameter to a sequence of corresponding multibyte characters and places the results in the area pointed to by the String parameter. The conversion is terminated when the null wide character is encountered or when the number of bytes specified by the Number parameter (or the value of the Number parameter minus 1) has been placed in the area pointed to by the String parameter. If the amount of space available in the area pointed to by the String parameter would cause a partial multibyte character to be stored, the subroutine uses a number of bytes equalling the value of the Number parameter minus 1, because only complete multibyte characters are allowed.
Parameters

String  Points to the area where the result of the conversion is stored. If the String parameter is a null pointer, the subroutine returns the number of bytes required to hold the conversion.

WcString  Points to a wide-character string.

Number  Specifies a number of bytes to be converted.

Return Values

The wcstombs subroutine returns the number of bytes modified. If a wide character is encountered that is not valid, a value of -1 is returned.

Error Codes

The wcstombs subroutine is unsuccessful if the following error occurs:

EILSEQ  An invalid character sequence is detected, or a wide-character code does not correspond to a valid character.

Related Information

The mbstowcs subroutine, mbtowc subroutine, wcslen subroutine, wcslen subroutine, wctomb subroutine.

wcstoul or wcstoull Subroutine

Purpose

Converts wide character strings to unsigned long or long long integer representation.

Library

Standard C Library (libc.a)

Syntax

#include <stdlib.h>

unsigned long int wcstoul (Nptr, Endptr, Base)
const wchar_t *Nptr;
wchar_t **Endptr;
int Base;

unsigned long long int wcstoull (Nptr, Endptr, Base)
const wchar_t *Nptr;
wchar_t **Endptr;
int Base;

Description

The wcstoul and wcstoull subroutines convert the initial portion of the wide character string pointed to by the Nptr parameter to an unsigned long or long long integer representation. To do this, it parses the wide character string pointed to by the Nptr parameter to obtain a valid string (that is, subject string) for the
purpose of conversion to an unsigned long integer. It then points the \textit{Endptr} parameter to the position where an unrecognized character, including the terminating null, is found.

The base specified by the \textit{Base} parameter can take the following values: 0 through 9, a (or A) through z (or Z). There are potentially 36 values for the base. If the base value is 0, the expected form of the subject string is that of an unsigned integer constant, with an optional + (plus sign) or - (minus sign), but not including the integer suffix. If the base value 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 the \textit{Base} parameter, optionally preceded by a + or -, but not including an integer suffix.

The letters a (or A) through z (or Z) are ascribed the values of 10 to 35. Only letters whose values are less than that of the base are permitted. If the value of the base is 16, the characters 0x (or 0X) may optionally precede the sequence of letters or digits, following a + or -. present.

The wide character string is parsed to skip the initial white-space characters (as determined by the \textit{iswspace} subroutine). Any nonspace character signifies the start of a subject string that may form an unsigned long integer in the radix specified by the \textit{Base} parameter. The subject sequence is defined to be the longest initial substring that is an unsigned long integer of the expected form. Any character not satisfying this expected form begins the final portion of the wide character string pointed to by the \textit{Endptr} parameter on return from the call to this subroutine.

\textbf{Parameters}

\textit{Nptr} Contains a pointer to the wide character string to be converted to an unsigned long integer.

\textit{Endptr} Contains a pointer to the position in the \textit{Nptr} string where a wide character is found that is not a valid character for the purpose of this conversion.

\textit{Base} Specifies the radix in which the wide characters are interpreted.

\textbf{Return Values}

The \textit{wcstoul} and \textit{wcstoull} subroutines return the converted value of the unsigned long or long long integer if the expected form is found. If no conversion could be performed, a value of 0 is returned. If the converted value is outside the range of representable values, a \texttt{ULONG\_MAX} value is returned (for \textit{wcstoul}), and \texttt{ULLONG\_MAX} is returned (for \textit{wcstoull}), and the value of the \textit{errno} global variable is set to a \texttt{ERANGE} value.

If the subject sequence has the expected form, it is interpreted as an integer constant in the appropriate base. A pointer to the final string is stored in the \textit{Endptr} parameter if that parameter is not a null pointer. If the subject sequence is empty or does not have a valid form, no conversion is done and the value of the \textit{Nptr} parameter is stored in the \textit{Endptr} parameter if it is not a null pointer.

If the radix specified by the \textit{Base} parameter is not supported, an \texttt{EINVAL} value is returned. If the value to be returned is not representable, an \texttt{ERANGE} value is returned.

\textbf{Examples}

To convert a wide character string to an unsigned long integer, use the following code:

```c
#include <stdlib.h>
#include <locale.h>
#include <errno.h>
extern int errno;

main()
{
    wchar_t *WCString, *EndPtr;
    unsigned long int retval;
```
(void)setlocale(LC_ALL, "");
/*
** Let WCString point to a wide character null terminated
** string containing an unsigned long integer value.
***/
retval = wcstoul (WCString &EndPtr, 0);
if(retval==0) {
  /* No conversion could be performed */
  /* Handle this case accordingly. */
} else if(retval == ULONG_MAX) {
  /* Error handling */
} /* retval contains the unsigned long integer value. */

Related Information

Subroutines, Example Programs, and Libraries in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs

wcswcs Subroutine

Purpose
Locates first occurrence of a wide character in a string.

Library
Standard C Library (libc.a)

Syntax
#include <string.h>

wchar_t *wcswcs(WcString1, WcString2)
const wchar_t *WcString1, *WcString2;

Description
The wcswcs subroutine locates the first occurrence, in the string pointed to by the WcString1 parameter, of a sequence of wchar_t characters (excluding the terminating wchar_t null character) from the string pointed to by the WcString2 parameter.

Parameters
WcString1 Points to the wide-character string being searched.
WcString2 Points to a wide-character string, which is a source string.

Return Values
The wcswcs subroutine returns a pointer to the located string, or a null value if the string is not found. If the WcString2 parameter points to a string with 0 length, the function returns the WcString1 value.
**Related Information**

The `mbspbrk` subroutine, `wcschr` subroutine, `wcscspn` subroutine, `wcspbrk` subroutine, `wcsrchr` subroutine, `wcsspn` subroutine, `wcstok` subroutine.


**wcswidth Subroutine**

**Purpose**
Determine the display width of wide character strings.

**Library**
Standard C Library (`libc.a`)

**Syntax**
```c
#include <string.h>

int wcswidth (const wchar_t *Pwcs, size_t n);
```

**Description**
The `wcswidth` subroutine determines the number of display columns to be occupied by the number of wide characters specified by the `N` parameter in the string pointed to by the `Pwcs` parameter. The `LC_CTYPE` category affects the behavior of the `wcswidth` subroutine. Fewer than the number of wide characters specified by the `N` parameter are counted if a null character is encountered first.

**Parameters**
- `N` Specifies the maximum number of wide characters whose display width is to be determined.
- `Pwcs` Contains a pointer to the wide character string.

**Return Values**
The `wcswidth` subroutine returns the number of display columns to be occupied by the number of wide characters (up to the terminating wide character null) specified by the `N` parameter (or fewer) in the string pointed to by the `Pwcs` parameter. A value of zero is returned if the `Pwcs` parameter is a wide character null pointer or a pointer to a wide character null (that is, `Pwcs` or `*Pwcs` is null). If the `Pwcs` parameter points to an unusable wide character code, -1 is returned.

**Examples**
To find the display column width of a wide character string, use the following:
```c
#include <string.h>
#include <locale.h>
#include <stdlib.h>
```
main()
{
    wchar_t *pwcs;
    int   retval, n ;
    (void)setlocale(LC_ALL, "");
    /* Let pwcs point to a wide character null terminated
       ** string. Let n be the number of wide characters whose
       ** display column width is to be determined.
       */
    retval= wcswidth( pwcs, n );
    if(retval == -1){
        /* Error handling. Invalid wide character code
           ** encountered in the wide character string pwcs.
           */
    }
}

Related Information
The **wcwidth subroutine**.

National Language Support Overview and Wide Character Display Column Width Subroutines in AIX 5L

Subroutines, Example Programs, and Libraries in AIX 5L Version 5.3 General Programming Concepts:
Writing and Debugging Programs.

wcsxfrm Subroutine

Purpose
Transforms wide-character strings to wide-character codes of current locale.

Library
Standard C Library (libc.a)

Syntax

```c
#include <string.h>

size_t wcsxfrm ( const wchar_t *WcString1, const wchar_t *WcString2, size_t Number);
```

Description
The **wcsxfrm** subroutine transforms the wide-character string specified by the **WcString2** parameter into a string of wide-character codes, based on the collation values of the wide characters in the current locale as specified by the **LC_COLLATE** category. No more than the number of character codes specified by the **Number** parameter are copied into the array specified by the **WcString1** parameter. When two such transformed wide-character strings are compared using the **wcsncmp** subroutine, the result is the same as that obtained by a direct call to the **wcsncmp** subroutine on the two original wide-character strings.

Parameters

- **WcString1** Points to the destination wide-character string.
- **WcString2** Points to the source wide-character string.
Number Specifies the maximum number of wide-character codes to place into the array specified by WcString1. To determine the necessary size specification, set the Number parameter to a value of 0, so that the WcString1 parameter becomes a null pointer. The return value plus 1 is the size necessary for the conversion.

Return Values
If the WcString1 parameter is a wide-character null pointer, the wcsxfrm subroutine returns the number of wide-character elements (not including the wide-character null terminator) required to store the transformed wide character string. If the count specified by the Number parameter is sufficient to hold the transformed string in the WcString1 parameter, including the wide character null terminator, the return value is set to the actual number of wide character elements placed in the WcString1 parameter, not including the wide character null. If the return value is equal to or greater than the value specified by the Number parameter, the contents of the array pointed to by the WcString1 parameter are indeterminate. This occurs whenever the Number value parameter is too small to hold the entire transformed string. If an error occurs, the wcsxfrm subroutine returns the size_t data type with a value of -1 and sets the errno global variable to indicate the error.

If the wide character string pointed to by the WcString2 parameter contains wide character codes outside the domain of the collating sequence defined by the current locale, the wcsxfrm subroutine returns a value of EINVAL.

Related Information
The wcsxfrm subroutine.

Return Values

The `wctob` function returns EOF if `c` does not correspond to a character with length one in the initial shift state. Otherwise, it returns the single-byte representation of that character.

Related Information

The `btowc` subroutine.

**wctomb Subroutine**

**Purpose**

Converts a wide character into a multibyte character.

**Library**

Standard C Library (`libc.a`)

**Syntax**

```c
#include <stdlib.h>

int wctomb ( Storage, WideCharacter )
char *Storage;
wchar_t WideCharacter;
```

**Description**

The `wctomb` subroutine determines the number of bytes required to represent the wide character specified by the `WideCharacter` parameter as the corresponding multibyte character. It then converts the `WideCharacter` value to a multibyte character and stores the results in the area pointed to by the `Storage` parameter. The `wctomb` subroutine can store a maximum of `MB_CUR_MAX` bytes in the area pointed to by the `Storage` parameter. Thus, the length of the area pointed to by the `Storage` parameter should be at least `MB_CUR_MAX` bytes. The `MB_CUR_MAX` macro is defined in the `stdlib.h` file.

**Parameters**

- `Storage` Points to an area where the result of the conversion is stored.
- `WideCharacter` Specifies a wide-character value.

**Return Values**

The `wctomb` subroutine returns a 0 if the `Storage` parameter is a null pointer. If the `WideCharacter` parameter does not correspond to a valid multibyte character, a -1 is returned. Otherwise, the number of bytes that comprise the multibyte character is returned.

**Related Information**

The `mbtowc` subroutine, `mbstowcs` subroutine, `wcslen ("wcslen Subroutine" on page 497)` subroutine, `wcstombs ("wcstombs Subroutine" on page 510)` subroutine.


Subroutines, Example Programs, and Libraries in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
wctrans Subroutine

Purpose
Define character mapping.

Library
Standard library (libc.a)

Syntax
#include <wctype.h>

wctrans_t *wctrans (const char *charclass);

Description
The wctrans function is defined for valid character mapping names identified in the current locale. The charclass is a string identifying a generic character mapping name for which codeset-specific information is required. The following character mapping names are defined in all locales “tolower” and “toupper”.

The function returns a value of type wctrans_t, which can be used as the second argument to subsequent calls of towctrans. The wctrans function determines values of wctrans_t according to the rules of the coded character set defined by character mapping information in the program’s locale (category LC_CTYPE). The values returned by wctrans are valid until a call to setlocale that modifies the category LC_CTYPE.

Return Values
The wctrans function returns 0 if the given character mapping name is not valid for the current locale (category LC_CTYPE), otherwise it returns a non-zero object of type wctrans_t that can be used in calls to towctrans.

Error Codes
The wctrans function may fail if:
EINVAL The character mapping name pointed to by charclass is not valid in the current locale.

Related Information
The towctrans function.

wctype or get_wctype Subroutine

Purpose
Obtains a handle for valid property names in the current locale for wide characters.

Library
Standard C library (libc.a).

Syntax
#include <wchar.h>
wctype_t wctype (const char *property);

wctype_t get_wctype (char *property);

Description

The wctype subroutine obtains a handle for valid property names for wide characters as defined in the current locale. The handle is of data type wctype_t and can be used as the WC_PROP parameter in the iswctype subroutine. Values returned by the wctype subroutine are valid until the setlocale subroutine modifies the LC_CTYPE category. The get_wctype subroutine is identical to the wctype subroutine.

The wctype subroutine adheres to X/Open Portability Guide Issue 5.

Parameters

Property
Points to a string that identifies a generic character class for which code set-specific information is required. The basic character classes are:

- alnum: Alphanumeric character.
- alpha: Alphabetic character.
- blank: Space and tab characters.
- cntrl: Control character. No characters in alpha or print are included.
- digit: Numeric digit character.
- graph: Graphic character for printing. Does not include the space character or cntrl characters, but does include all characters in digit and punct.
- lower: Lowercase character. No characters in cntrl, digit, punct, or space are included.
- print: Print character. Includes characters in graph, but does not include characters in cntrl.
- punct: Punctuation character. No characters in alpha, digit, or cntrl, or the space character are included.
- space: Space characters.
- upper: Uppercase character.
- xdigit: Hexadecimal character.

Return Values

Upon successful completion, the subroutine returns a value of type wctype_t, which is a handle for valid property names in the current locale. Otherwise, it returns a value or -1 if the Property parameter specifies a character class that is not valid for the current locale.

Related Information

The iswalnum subroutine, iswalpha subroutine, iswcntrl subroutine, iswctype subroutine, iswdigit subroutine, iswgraph subroutine, iswlower subroutine, iswprint subroutine, iswpunct subroutine, iswspace subroutine, iswupper subroutine, iswxdigit subroutine, setlocale subroutine, towlower subroutine, towupper subroutine.

National Language Support Overview | Wide Character Classification Subroutines in AIX 5L Version 5.3
**wcwidth Subroutine**

**Purpose**
Determines the display width of wide characters.

**Library**
Standard C Library (libc.a)

**Syntax**
```c
#include <string.h>

int wcwidth ( wchar_t WC);
```

**Description**
The `wcwidth` subroutine determines the number of display columns to be occupied by the wide character specified by the `WC` parameter. The `LC_CTYPE` subroutine affects the behavior of the `wcwidth` subroutine.

**Parameters**

`WC`  Specifies a wide character.

**Return Values**
The `wcwidth` subroutine returns the number of display columns to be occupied by the `WC` parameter. If the `WC` parameter is a wide character null, a value of 0 is returned. If the `WC` parameter points to an unusable wide character code, -1 is returned.

**Examples**
To find the display column width of a wide character, use the following:
```c
#include <string.h>
#include <locale.h>
#include <stdlib.h>

main()
{
    wchar_t wc;
    int _retval;
    (void)setlocale(LC_ALL, "");
    /* Let wc be the wide character whose ** display width is to be found. */
    _retval= wcwidth( wc );
    if(_retval == -1)
    { /*
          ** Error handling. Invalid wide character in wc. */
    }
}
```
Related Information
The `wcswidth` subroutine.


Subroutines, Example Programs, and Libraries in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

wlm_assign Subroutine

Purpose
Manually assigns processes to a class or cancels prior manual assignments for processes.

Library
Workload Manager Library (libwlm.a)

Syntax
```c
#include <sys/wlm.h>

int wlm_assign ( struct wlm_assign* args );
```

Description
The `wlm_assign` subroutine:

- Assigns a set of processes specified by their process IDs (PIDS) or process group IDs (PGID) to a specified superclass or subclass, thus overriding the automatic class assignment or a prior manual assignment.
- Cancels a previous manual assignment for the specified processes, allowing the processes to be subjected to the automatic assignment rules again.

The target processes are identified by their process ID (pid) or by their process group ID (pgid). The `wlm_assign` subroutine allows specifying processes using a list of pids, a list of pgids, or both.

The name of a valid superclass or subclass must be specified to manually assign the target processes to a class. If the target class is a superclass, each process is assigned to one of the subclasses of the specified superclass according to the assignment rules for the subclasses of this superclass.

A manual assignment remains in effect (and a process remains in its manually assigned class) until:

- The process terminates.
- The Workload Manager (WLM) is stopped. When WLM is restarted, the manual assignments in effect when WLM was stopped are lost.
- The class the process has been assigned to is deleted.
- The manual assignment for the process is canceled.
- A new manual assignment overrides a prior one.

The name of a valid superclass or subclass must be specified to manually assign the target processes to a class. The assignment can be done or canceled at the superclass level, the subclass level, or both. The
interactions between automatic assignment, inheritance and manual assignment are detailed in the Manual Assignment in WLM in AIX 5L Version 5.3 System Management Concepts: Operating System and Devices.

Flags in the wa_versflags field described below are used to specify if the requested operation is an assignment or cancellation and at which level.

To assign a process to a class or cancel a prior manual assignment, the caller must have authority both on the process and on the target class. These constraints translate into the following:

- The root user can assign any process to any class.
- A user with administration privileges on the subclasses of a given superclass (that is, the user or group name matches the user or group names specified in the attributes adminuser and admingroup of the superclass) can manually reassign any process from one of the subclasses of this superclass to another subclass of the superclass.
- A user can manually assign the user’s own processes (same real or effective user ID) to a superclass or a subclass, for which the user has manual assignment privileges (that is, the user or group name matches the user or group names specified in the attributes authuser and authgroup of the superclass or the subclass).

This defines three levels of privilege among the persons who can manually assign processes to classes, root being the highest. For a user to modify or terminate a manual assignment, the user must be at the same level of privilege as the person who issued the last manual assignment, or higher.

Note: The wlm_assign subroutine works with the in-core WLM data structures. Even if the WLM current configuration is a set, it applies to the currently loaded regular configuration. If an assignment is made to a class that does not exist in all configurations of the set, it will be lost when the first configuration that does not contain this class is activated (when the class is deleted).

Parameter

args Specifies the address of the struct wlm_assign data structure containing the parameters for the desired class assignment.

The following fields of the wlm_args structure and the embedded substructures can be provided:

wa_versflags Needs to be initialized with WLM_VERSION. The flags values available, defined in the sys/wlm.h header file, are:
- WLM_ASSIGN_SUPER
- WLM_ASSIGN_SUB
- WLM_ASSIGN_BOTH
- WLM_UNASSIGN_SUPER
- WLM_UNASSIGN_SUB
- WLM_UNASSIGN_BOTH

wa_pids Specifies the address of the array containing the process IDs of processes to be manually assigned. When this list is empty, a NULL pointer can be passed together with a count of zero (0).

wa_pid_count Specifies the number of PIDS in the above array. Could be zero (0) if using only pgids to identify the processes.
wa_pgid_count

Specifies the number of PGIDs in the above array. Could be zero (0) if using only pids to identify the processes. If both pids and pgids counts are zero (0), no process is assigned, but the operation is considered successful.

wa_classname

Specifies the full name of the superclass (super_name) or the subclass (super_name.sub_name) of the class you want to manually assign processes to. The class name field is ignored when canceling an existing manual assignment.

Return Values

Upon successful completion, the wlm_assign subroutine returns a value of 0. If the wlm_assign subroutine is unsuccessful, a non-0 value is returned. The routine is considered successful if some of the target processes are not found, (to account for process terminations) or are not assigned/deassigned due to a lack of privileges, for instance. If none of the processes in the lists can be assigned/deassigned, this is considered an error.

Error Codes

For a list of the possible error codes returned by the WLM API functions, see the description of the wlm.h header file.

Related Information


wlm_change_class Subroutine

Purpose

Changes some of the attributes of a class.

Library

Workload Manager Library (libwlm.a)

Syntax

#include <sys/wlm.h>

int wlm_change_class ( wlmargs)

struct wlm_args *wlmargs;

Description

The wlm_change_class subroutine changes attributes of an existing superclass or subclass. Except for its name, any of the attributes of the class can be modified by a call to wlm_change_class.

- If the name of a valid configuration is passed in the confdir field, the subroutine updates the Workload Manager (WLM) properties files for the target configuration.
- If a null string ('\0') is passed in the confdir field, the changes are applied only to the in-core WLM data. No WLM properties file is updated.
The structure of type `struct class_definition`, which is part of `struct wlm_args`, has normally been initialized with a call to `wlm_init_class_definition`. Once this has been done, initialize the required fields of this structure (such as the name of the class to be modified) and the fields corresponding to the class attributes you want to modify. For a description of the possible values for the various class attributes and their default values, refer to the description of `wlm.h` in the AIX 5L Version 5.3 Files Reference.

The caller must have root authority to change the attributes of a superclass and must have administrator authority on a superclass to change the attributes of a subclass of the superclass.

**Note:** Do not specify a set in the `confdir` field of the `wlm_args` structure. The `wlm_change_class` subroutine cannot apply to a set of time-based configurations.

**Parameters**

- **wlmargs**
  Specifies the address of the `struct wlm_args` data structure containing the `class_definition` structure for the class to be modified.

The following fields of the `wlm_args` structure and the embedded substructures need to be provided:

- **versflags**
  Needs to be initialized with `WLM_VERSION`.

- **confdir**
  Specifies the name of the WLM configuration the target class belongs to. It must be either the name of a valid subdirectory of `/etc/wlm` or an empty string (starting with `"0"`).

  If the name is a valid subdirectory, the relevant class description file in the given configuration are modified.

  If the name is a null string, no description files are updated. The modified class attributes are passed to the kernel similarly to a call to `wlm_load`.

- **name**
  Specifies the name of the superclass or of the subclass to be modified. If this is a subclass name, it must be of the form `super_name.sub_name`. There is no default for this field.

All the other fields can be left at their initial value as set by `wlm_init_class_definition` if the user does not wish to change the current values.

**Return Values**

Upon successful completion, the `wlm_change_class` subroutine returns a value of 0. If the `wlm_change_class` subroutine is unsuccessful, a nonzero value is returned.

**Error Codes**

For a list of the possible error codes returned by the WLM API functions, see the description of the `wlm.h` header file.

**Related Information**

The `wlm.h` header file.

The `wlm_create_class` subroutine, `wlm_delete_class` subroutine.

For more information, refer to the Workload Manager Application Programming Interface in AIX 5L Version 5.3 System Management Concepts: Operating System and Devices.
wlm_check subroutine

Purpose
Check a WLM configuration.

Library
Workload Manager Library (libwlm.a)

Syntax
#include <sys/wlm.h>

int wlm_check ( config)
char *config;

Description
The wlm_check subroutine checks the class definitions and the coherency of the assignment rules file(s) (syntax, existence of the classes, validity of user and group names, application path names, etc.) for the configuration whose name is passed as an argument.

If config is a null pointer or points to an empty string, wlm_check performs the checks on the configuration files, in the configuration pointed to by /etc/wlm/current.

The wlm_check subroutine can apply to a configuration set. If config is a configuration set name (or if config is not provided and current is a configuration set), the checks mentioned above are performed on all configurations of the set, after checking the set itself.

Parameter
config A pointer to a character string. This pointer should be:
- The address of a character string representing the name of a valid configuration (a subdirectory of /etc/wlm)
- A null pointer
- A pointer to a null string ("")

If config is a null pointer or a pointer to a null string, the configuration files in the directory pointed to by /etc/wlm/current (active configuration) is checked for errors. Otherwise, the configuration files in directory /etc/wlm/<config_name> is checked.

Return Values
Upon successful completion, a value of 0 is returned. If the wlm_check subroutine is unsuccessful a non 0 value is returned.

Error Codes
For a list of the possible error codes returned by the WLM API functions, see the description of the header file sys/wlm.h.

Related Information
The wlm.h header file.
The rules file.

**wlm_classify Subroutine**

**Purpose**
Determines which classes a process is assigned to.

**Library**
Workload Manager Library (libwlm.a)

**Syntax**
```c
#include <sys/wlm.h>

int wlm_classify ( config, attributes, class, len);
```

```c
char *config;
char *attributes;
char *class;
int *len;
```

**Description**
The `wlm_classify` subroutine must be passed the name of a valid configuration and a set of process attributes in a format identical to the format of the rules file (assignment rules). The names of the classes are copied into the area pointed to by class. The integer pointed to by len contains the size of the class names area on input and the number of matches on output. If the area pointed to by class is not big enough to contain the names of all the potential matches, an error is returned.

The normal use of the `wlm_classify` routine is to explicitly provide all the process classification attributes: user name, group name, application pathname, type, and tag when applicable. This gives a match to a single class. To implement "what if" scenarios, the interface allows you to leave some of the attributes unspecified by using a hyphen ("-") instead. This may lead to multiple classes the process could be assigned to, depending on the values of the unspecified attributes. If all the attributes are left unspecified, an error is returned.

The attributes string is provided in a format identical to the format of the attributes in the rules file: a list of attribute values separated by spaces. The order of the attributes in the assignment rules is:
1. reserved: must be a hyphen ("-"
2. user name
3. group name
4. application pathname
5. type of application
6. tag

Each field can have at most one value. Exclusion (!), attribute value groupings ($), comma separated lists and wild cards are not allowed. For the type field, the AND operator "+" is allowed, since a process can
have several of the possible values for the type attribute at the same time. For instance a process can be a 32 bit process and call plock, or be a 64 bit fixed priority process.

Here are examples of valid attributes strings:
"- bob staff /usr/bin/emacs - -"
"- - - /usr/sbin/dbserv - _DB1"
"- - devlt - 32bit+fixed"
"- - sally"

The class name(s) returned by the function in the class buffer is fully-qualified, null-terminated class names of the form supername.subname.

This function does not require any special privileges and can be called by all users.

Parameters

config Specifies a pointer to a string containing the name of a valid Workload Manager (WLM) configuration (the name of a subdirectory of /etc/wlm). If a null string ("\0") is given, the wlm_classify subroutine uses current as the default configuration.

If the configuration is a set of time-based configurations, either because config or current is a configuration set, the subroutine will apply to the currently applicable configurations of the set.

attributes Specifies the address of a string, with the format described above, containing a list of values for the process attributes used for automatic classification of processes.

class Specifies a pointer to a buffer where the name of the class the process could be assigned to is returned as consecutive null-terminated character strings.

len Specifies a pointer to an integer containing the length in bytes of the buffer pointed to by class when calling wlm_classify and the actual number of class names copied into the class buffer upon successful return.

Return Values

Upon successful completion, the wlm_classify subroutine returns a value of 0. In case of error, a non-0 value is returned.

When a non-0 value is returned, the content of the class buffer and the value of the integer pointed to by len are unspecified.

Error Codes

For a list of the possible error codes returned by the WLM API functions, see the description of the wlm.h header file.

Related Information

The wlmcheck command.

The wlm.h header file.

Workload Manager rules File in AIX 5L Version 5.3 Files Reference.

wlm_class2key Subroutine

Purpose
Class name to key translation.

Library
Workload Manager Library (libwlm.a)

Syntax
#include <sys/wlm.h>

int wlm_class2key (struct wlm_args *args, wlm_key_t *key)

Description
The wlm_class2key subroutine generates a 64-bit numeric key from a WLM class name. The wlm_class2key subroutine is provided for applications gathering high volumes of per-class usage statistics or accounting data and allows those applications to save storage space by compressing the class name (up to 34 characters long) into a 64-bit integer. The wlm_key2class subroutine can then get the key-to-class name conversion for data reporting purposes.

Parameters
wlm_args
Only 2 fields need to be initialized in the wlm_args structure pointed to by args:
- cl_def.data.descr.name specifies the null terminated full name of the class (<super_name>.<subname> for a subclass).
- versflags initialized with WLM_VERSION and optionally WLM_MUTE.

Return Values
If the wlm_class2key subroutine is successful, a value of 0 is returned. If the wlm_class2key subroutine is unsuccessful, an error code is returned.

Error Codes
If the wlm_class2key subroutine is unsuccessful, one of the following error codes is returned:
- WLM_NOT_INITED Missing call to wlm_init.
- WLM_EFAULT Invalid key or args pointer.
- WLM_BADCNAME The class name contains invalid characters.

Related Information
The wlm_endkey subroutine.
The wlm_initkey subroutine.
The wlm_key2class subroutine.

wlm_create_class Subroutine

Purpose
Creates a new Workload Manager (WLM) class.
Library
Workload Manager Library (libwlm.a)

Syntax
#include <sys/wlm.h>

int wlm_create_class (wlmargs);

struct wlm_args *wlmargs;

Description
The wlm_create_class subroutine creates a new class for a given WLM configuration using the values passed in the data structure of type struct wlm_args pointed to by wlmargs.

- If the name of a configuration is passed in the confdir field, the subroutine updates the WLM properties files for the target configuration. When creating the first subclass of a superclass, the subroutine creates a subdirectory of /etc/wlm/<confdir> with the name of the superclass and create the WLM properties files in this new directory. The newly created properties files have entries for the Default and Shared subclass automatically created in addition to entries for the new subclass.
- If a null string ('\0') is passed in the confdir field, the new superclass or subclass is created only in the in-core WLM data. No WLM properties file are updated. In that case, the new class definition is lost if WLM is stopped and restarted, or if the system reboots.

The structure of type struct class_definition, which is part of struct wlm_args, has normally been initialized with a call to wlm_init_class_definition. Once this has been done, initialize the fields of this structure which have no default value (such as the name of the new class) or for which the desired value is different from the default value. For a description of the possible values for all the class attributes and their default values, refer to the description of wlm.h in the AIX 5L Version 5.3 Files Reference.

The caller must have root authority to create a superclass and must have administrator authority on a superclass to create a subclass of the superclass.

Note: Do not specify a set in the confdir field of the wlm_args structure. The wlm_create_class subroutine cannot apply to a set of time-based configurations.

Parameter

wlmargs Specifies the address of the struct wlm_args data structure containing the class_definition structure for the new class to be created.

The following fields of the wlm_args structure and the embedded substructures need to be provided:

versflags Needs to be initialized with WLM_VERSION.
confdir Specifies the name of the WLM configuration the new class is to be added to. It must be either the name of a valid subdirectory of /etc/wlm or an empty string (starting with '\0').

If the name is a valid subdirectory, the new class data is added to the given WLM configuration’s class description files.

If the name is a null string, no description files are updated. The new class is created and the data is passed to the kernel immediately.

name Specifies the name of the superclass or of the subclass to be created. If this is a subclass name, it must be of the form super_name.sub_name. There is no default for this field.

All the other fields can be left at their default value if the user does not wish to use specific values.
Return Values
Upon successful completion, the `wlm_create_class` subroutine returns a value of 0. If the `wlm_create_class` subroutine is unsuccessful, a nonzero value is returned.

Error Codes
For a list of the possible error codes returned by the WLM API functions, see the description of the `wlm.h` header file.

Related Information
The `mkclass` command, `chclass` command, `rmclass` command.

The `wlm.h` header file.

The `wlm_change_class` subroutine, `wlm_delete_class` subroutine.

`Workload Management in AIX 5L Version 5.3 System Management Concepts: Operating System and Devices`

---

**wlm_delete_class Subroutine**

**Purpose**
Deletes a class.

**Library**
Workload Manager Library (`libwlm.a`)

**Syntax**
```
#include <sys/wlm.h>

int wlm_delete_class (wlmargs)

struct wlm_args *wlmargs;
```

**Description**
The `wlm_delete_class` subroutine deletes an existing superclass or subclass. A superclass cannot be deleted if it still has subclasses other than Default and Shared defined.

- If the name of a valid configuration is passed in the `confdir` field, the subroutine updates the Workload Manager (WLM) properties files for the target configuration, removing all references to the class to be deleted.
- If a null string (`"\0"`) is passed in the `confdir` field, the class is deleted only from the in-core WLM data structures. No WLM properties file is updated. This is normally used to delete a class which was also only created in the in-core WLM data structures. Otherwise, the class deletion is temporary and the class will be created again when WLM is updated or restarted with a configuration where the class exists in the classes file.

The caller must have root authority to delete a superclass and must have administrator authority on a superclass to delete a subclass of the superclass.

**Note:** Do not specify a set in the `confdir` field of the `wlm_args` structure. The `wlm_delete_class` subroutine cannot apply to a set of time-based configurations.
Parameter

wlmargs

Specifies the address of the struct wlm_args data structure containing the information about the class to be deleted.

The following fields of the wlm_args structure and the embedded substructures need to be provided:

- versflags
  Needs to be initialized with WLM_VERSION.

- confdir
  Specifies the name of the WLM configuration the target class belongs to. It must be either the name of a valid subdirectory of /etc/wlm or an empty string (starting with '\0').

  If the name is a valid subdirectory, the relevant class description files in the specified configuration are modified.

  If the name is a null string, no description files are updated. The class is removed from the kernel WLM data structures.

- name
  Specifies the name of the superclass or of the subclass to be deleted. If this is a subclass name, it must be of the form super_name.sub_name. There is no default for this field.

  All the other fields can be left uninitialized for this call.

Return Values

Upon successful completion, the wlm_delete_class subroutine returns a value of 0. If the wlm_delete_class subroutine is unsuccessful, a non-0 value is returned.

Error Codes

For a list of the possible error codes returned by the WLM API functions, see the description of the wlm.h header file.

Related Information

The mkclass command, chclass command, rmclass command.

The wlm.h header file.

The wlm_change_class subroutine, wlm_create_class subroutine.

Workload Management in AIX 5L Version 5.3 System Management Concepts: Operating System and Devices.

wlm_endkey Subroutine

Purpose

Frees the classes to keys translation table.

Library

Workload Manager Library (libwlm.a)
**Syntax**
```c
#include <sys/wlm.h>

int wlm_endkey(struct wlm_args *args, void *ctx)
```

**Description**
The `wlm_endkey` subroutine frees the classes to the keys translation table. The memory area pointed to by `ctx` is freed.

**Parameters**
- `ctx` Points to the memory area to be freed.
- `wlm_args` A pointer to a `wlm_args` structure:
  - `versflag` field is the only field in the structure that needs to be initialized with `WLM_VERSION` and optionally `WLM_MUTE`.

**Return Values**
When the `wlm_endkey` operation is successful, it returns a value of 0, and if it is unsuccessful, it returns an error code.

**Error Codes**
If the `wlm_endkey` subroutine is unsuccessful, one of the following error codes is returned:

- `WLM_BADVERS` Bad version number.
- `WLM_NOT_INITED` Missing call to `wlm_init`.
- `WLM_EFAULT` Invalid `ctx` or `args` argument.

**Related Information**
The `wlm_class2key` subroutine.
The `wlm_initkey` subroutine.
The `wlm_key2class` subroutine.

---

**wlm_get_bio_stats subroutine**

**Purpose**
Read the WLM disk I/O statistics per class or per device.

**Library**
Workload Manager Library (`libwlm.a`)

**Syntax**
```c
#include <sys/types.h>
#include <sys/wlm.h>

int wlm_get_bio_stats (dev, array, count, class, flags)
```
dev_t dev;
void *array;
int *count;
char *class;
int flags;

Description
The wlm_get_bio_stats subroutine is used to get the WLM disk IO statistics. There are two types of statistics available:

- The statistics about disk IO utilization per class and per devices, returned by wlm_get_bio_stats in wlm_bio_class_info_t structures,
- The statistics about the disk IO utilization per device, all classes combined, returned by wlm_get_bio_stats in wlm_bio_dev_info_t structures.

The type of statistics returned by the function is predicated on the value of the flags argument. The flags argument, together with the dev and class arguments, are used to restrict the scope of the function to a class or a set of classes and/or a device or a set of devices. If the value passed to the routine in the count argument is equal to zero (0), wlm_get_bio_stats does not copy any device statistics (and, in this case, the array argument can be a NULL pointer but sets this count to the number of elements in scope for the specific set of parameters. This is a way of finding out how big an array is needed to get all the information for a given set of classes and devices.

wlm_get_bio_stats does not require any special privileges and is accessible to all users. wlm_get_bio_stats fails if WLM is off.
Parameters

flags

Need to be initialized with WLM_VERSION. Optionally, the following flag values can be or’ed to WLM_VERSION:

- **WLM_SUPER_ONLY**
  - Limits the scope to superclasses only

- **WLM_SUB_ONLY**
  - Limits the scope to subclasses only

- **WLM_BIO_CLASS_INFO**
  - Per class statistics requested

- **WLM_BIO_DEV_INFO**
  - Per device statistics requested

- **WLM_BIO_ALL_DEV**
  - Requests statistics for all devices. When this flag is set, the value passed in the dev argument is ignored.

- **WLM_BIO_ALL_MINOR**
  - Requests statistics for all devices associated with a given major number. When this flag is set, only the major number part of the value passed in the dev argument is used.

- **WLM_VERBOSE_MODE**
  - Shows the system defined subclasses (Default and Shared) even if they have not been modified by a WLM administrator.

One of the flags WLM_BIO_CLASS_INFO or WLM_BIO_DEV_INFO (and only one) must be specified. WLM_SUPER_ONLY and WLM_SUB_ONLY are mutually exclusive.

dev

Device identification (major, minor) of a disk device.

- If dev is equal to 0, the statistics for all devices are returned (even if WLM_BIO_ALL_DEV is not specified in the flags argument).

- If dev is not equal to 0 and WLM_BIO_ALL_MINOR is specified in the flags argument, the statistics for all disk devices with the same major number specified in dev are returned.

- If dev is not equal to 0 and WLM_BIO_ALL_MINOR is not specified in the flags argument, only the statistics for the disk device with the major and minor numbers specified in dev are returned.

array

Pointer to an array of wlm_bio_class_info_t structures (when WLM_BIO_CLASS_INFO is specified in the flags argument) or an array of wlm_bio_dev_info_t structures (when WLM_BIO_DEV_INFO is specified in the flags argument). A NULL pointer can be passed together with a count of 0 to determine how many elements are in scope for the set of arguments passed.

count

The address of an integer containing the maximum number of elements to be copied into the array above. If the call to wlm_get_bio_stats is successful, this integer will contain the number of elements actually copied. If the initial value is equal to zero (0), wlm_get_bio_stats sets this value to the number elements selected by the specified combination of flags and class.
A pointer to a character string containing the name of a superclass or subclass. If class is a pointer to an empty string (″″), the information for all classes are returned. The class parameter is taken into account only when the flag WLM_BIO_CLASS_INFO is set.

Return Values
Upon successful completion, a value of 0 is returned and the value pointed to by count is set to the number of elements copied into the array of structures pointed to by array. If the wlm_get_bio_stats subroutine is unsuccessful a non 0 value is returned.

Error Codes
For a list of the possible error codes returned by the WLM API functions, see the description of the header file sys/wlm.h.

Related Information
The wlm.h header file.

wlm_get_info Subroutine

Purpose
Read the characteristics of superclasses or subclasses.

Library
Workload Manager Library (libwlm.a)

Syntax
#include <sys/wlm.h>

int wlm_get_info ( wlmargs, info, count)
struct wlm_args *wlmargs;
struct wlm_info *info
int *count

Description
The wlm_get_info subroutine is used to get the characteristics of the classes defined in the active Workload Manager (WLM) configuration, together with their current resource usage statistics. For a detailed description of the fields of the structure wlm_info, refer to the description of the header file in the AIX 5L Version 5.3 Files Reference documentation.

By default, the scope of the wlm_get_info subroutine is all the superclasses and all the subclasses. This scope can be limited to a subset of the classes using flags in the versflags field of wlm_args or a superclass or subclass name in the name field of the substructure class_definition of wlm_args.

The information related to the superclasses and subclasses within the scope of wlm_get_info are copied to the array of wlm_info structures pointed to by info. The total number of classes for which information is copied to the array at info is limited to the value of the integer pointed to by count. If the routine is successful, the value of the integer pointed to by count is set to the actual number of classes copied. If the
value passed to the routine for the count is equal to zero (0), \texttt{wlm\_get\_info} does not copy any class statistics but sets this count to the number of classes in scope for the specific set of parameters. This is a way of finding out how big an array is needed to get all the information for a given set of classes (superclasses or subclasses).

This is a way of finding out how big an array is needed to get all the information for a given set of classes (superclasses or subclasses).

The \texttt{wlm\_get\_info} subroutine does not require any special privileges and is accessible to all users. \texttt{wlm\_get\_info} fails if WLM is off.

**Parameters**

\texttt{wlmargs}

The address of a \texttt{struct wlm\_args} data structure.

The following fields of the \texttt{wlm\_args} structure and the embedded substructures need to be provided:

\texttt{versflags}

Needs to be initialized with \texttt{WLM\_VERSION}. Optionally, the following flag value can be or\'ed to \texttt{WLM\_VERSION}:

\texttt{WLM\_SUPER\_ONLY}

Limits the scope to superclasses only

\texttt{WLM\_SUB\_ONLY}

Limits the scope to subclasses only

\texttt{WLM\_VERBOSE\_MODE}

Shows the system-defined subclasses (Default and Shared) even if they have not been modified by a WLM administrator.

\texttt{WLM\_SUPER\_ONLY} and \texttt{WLM\_SUB\_ONLY} are mutually exclusive.

\texttt{name}

Contains either a null string or the name of a valid superclass or subclass (in the form \texttt{Super.Sub}). This field can be used in conjunction with the flags to further narrow the scope of \texttt{wlm\_get\_info}:

- If the name of a subclass is provided, \texttt{wlm\_get\_info} returns the statistics only for the specified subclass.
- If the name of a superclass is provided or if none of the \texttt{WLM\_SUPER\_ONLY} and \texttt{WLM\_SUB\_ONLY} flag is provided, \texttt{wlm\_get\_info} returns the statistics for the specified superclass and all its subclasses.
- If the name of a superclass is provided together with \texttt{WLM\_SUPER\_ONLY}, \texttt{wlm\_get\_info} returns only the statistics for the specified superclass.
- If the name of a superclass is provided together with \texttt{WLM\_SUB\_ONLY}, \texttt{wlm\_get\_info} returns the statistics for all the subclasses of the specified superclass.

All the other fields of the \texttt{wlm\_args} structure can be left uninitialized.

\texttt{info}

The address of an array of structures of type \texttt{struct wlm\_info}. Upon successful return from \texttt{wlm\_get\_info}, this array contains the WLM statistics for the classes selected.

\texttt{count}

The address of an integer containing the maximum number of element (of type \texttt{wlm\_info}) for \texttt{wlm\_get\_info} to copy into the array above. If the call to \texttt{wlm\_get\_info} is successful, this integer contains the number of elements actually copied. If the initial value is equal to zero (0), \texttt{wlm\_get\_info} sets this value to the number of classes selected by the specified combination of \texttt{versflags} and \texttt{name} above.
Return Values
Upon successful completion, the `wlm_get_info` subroutine returns a value of 0. If the `wlm_get_info` subroutine is unsuccessful a non-0 value is returned.

Error Codes
For a list of the possible error codes returned by the WLM API functions, see the description of the `wlm.h` header file.

Related Information
The `wlmstat` command.
The `wlm.h` header file.

### wlm_get_procinfo Subroutine

**Purpose**
Retreives per-process Workload Manager information.

**Library**
Workload Manager Library (`libwlm.a`)

**Syntax**
```c
#include <sys/wlm.h>

int wlm_get_procinfo (pid, wlmpinfop)
  pid_t pid;
  struct wlm_procinfo *wlmpinfop;
```

**Description**
The `wlm_get_procinfo` subroutine returns Workload Manager information for the process associated with the `pid` parameter, into the buffer pointed to by the `wlmpinfop` parameter. If process total accounting is disabled, the related fields (`totalconnecttime`, `termtime`, `totalcputime`, and `totaldiskio`) are set to -1. When WLM is on, the class name of the process is set in the `classname` field of the `wlm_procinfo` structure. When WLM is off, this field is set to `Unclassified`.

**Parameters**
- `pid` Indicates from which process to retrieve the Workload Manager information.
- `wlmpinfop` Points to the buffer where the Workload Manager information is stored.

**Return Values**
Upon successful completion, the `wlm_get_procinfo` subroutine returns a zero. If the `wlm_get_procinfo` subroutine is unsuccessful, a nonzero value is returned.

**Error Codes**
For a list of the possible error codes returned by the WLM API functions, see the description of the `wlm.h` header file.

**Related Information**
The `wlm.h` header file.
**wlm_init_class_definition Subroutine**

**Purpose**
Initializes a variable of type `struct class_definition`, defined in `<sys/wlm.h>` for use as an argument to Workload Manager (WLM) API function calls.

**Library**
Workload Manager Library (`libwlm.a`)

**Syntax**
```
#include <sys/wlm.h>

int wlm_init_class_definition (wlmargs);
struct wlm_args *wlmargs;
```

**Description**
The `wlm_init_class_definition` subroutine initializes or reinitializes the data structure of type `struct class_definition`, which is part of the argument of type `struct wlm_args` pointed to by `wlmargs` (field `class`), so that this data structure can be used as an argument for the class management subroutines of the WLM API library. The purpose of this call is to allow applications to initialize only the fields that are relevant for the operation they execute. For example, to change a CPU limit or share for an existing class after a call to `wlm_init_class_definition`, the application has to initialize the fields corresponding to the values it wishes to modify.

This routine initializes all values to specific invalid values so that the WLM library routines can find out which fields have been explicitly initialized by the user. This way, they can set or modify only the corresponding attributes. When creating a class, for instance, it is different to leave a `class` attribute at its invalid value set by `wlm_initialize` than setting its value to the current default value for the attribute. In the former case, the attribute will not appear in the property file. In the latter, it will appear and will be set with the value passed.

This makes a difference if a WLM administrator decides to change the default value for an attribute using the special stanza default in a property file. For instance, the system default for the `inheritance` attribute is no. If a WLM administrator wants the inheritance to be yes by default, using this special stanza, all the classes in the classes property file, for which the `inheritance` attribute has not been specified, will now use the default of yes. Those for which the `inheritance` attribute has been specified with its old default of no will not have inheritance.

**Parameter**

- **wlmargs**
  Specifies the address of the `struct wlm_args` data structure containing the `class_definition` structure to be initialized.

Only the `versflags` field of the `wlm_args` structure passed need to be initialized with `WLM_VERSION`.

**Return Values**
Upon successful completion, the `wlm_init_class_definition` subroutine returns a value of 0. If the `wlm_init_class_definition` subroutine is unsuccessful a non-0 value is returned.
Error Codes
There are two possible error code returned by `wlm_init_class_definition`:

BADVERSION Specifies the value of the flags parameter is not a supported version number.

NOTINITED Specifies the WLM API has not been initialized by a prior call to `wlm_init`.

Related Information
The `wlm.h` header file.

The `wlm_change_class` subroutine, `wlm_create_class` subroutine, `wlm_delete_class` subroutine.

wlm_initialize Subroutine

Purpose
Prepares Workload Manager (WLM) for use by an application.

Library
Workload Manager Library (libwlm.a)

Syntax
```c
#include <sys/wlm.h>

int wlm_initialize(int flags);

int flags;
```

Description
The `wlm_initialize` subroutine initializes the WLM API for use with an application program. It is mandatory to call `wlm_initialize` prior to using the WLM API. Otherwise, all other WLM API function calls return an error.

Parameter
flags

Specifies that the format is the same as the versflag field of the `wlm_args` structure. The value for the argument must have the version number in the upper 4 bits (`WLM_VERSION`) possibly or’ed with a flag in the lower 28 bits.

Return Values
Upon successful completion, the `wlm_initialize` subroutine returns a value of 0. If the `wlm_initialize` subroutine is unsuccessful a non-0 value is returned.
Error Codes
There are two possible error codes returned by `wlm_initialize`:

**BADVERSION**  The value of the `flags` parameter is not a supported version number.
**WLMINITED**   There has already been a previous call to `wlm_initialize`.

Related Information
The `wlm.h` header file.

---

**wlm_initkey Subroutine**

**Purpose**
Allocates and initializes the classes to keys translation table.

**Library**
Workload Manager Library (`libwlm.a`)

**Syntax**
```c
#include <sys/wlm.h>

int wlm_initkey ( struct wlm_args *args, void **ctx)
```

**Description**
The `wlm_initkey` subroutine allocates a block of memory, builds the keys <--> class names translation table and returns its address into the `ctx` argument.

**Parameters**

- `args`  Only 2 fields need to be initialized in the `wlm_args` structure pointed to by `args`:
  - `confdir` specifies the null-terminated name of the WLM configuration to be searched (the name can be "current" to specify the current configuration). If the configuration name passed is an empty string (starts with '\'0"), then all the configurations in `/etc/wlm` are searched.
  - `versflags` initialized with WLM_VERSION and optionally WLM_MUTE.

**Return Values**
If the `wlm_initkey` subroutine is successful, a value of 0 is returned. If the `wlm_initkey` subroutine is unsuccessful, an error code is returned.

**Error Codes**
If the `wlm_initkey` subroutine is unsuccessful, one of the following error codes is returned:

- `WLM_BADVERS` Bad version number.
- `WLM_NOT_INITED` Missing call to `wlm_init`.
- `WLM_NOMEM` Not enough memory.
- `WLM_NOCLASS` Specified configuration does not exist.
- `WLM_EFAULT` Invalid `ctx` or `args` argument.
wlm_key2class Subroutine

Purpose
Retrieves a class name from a key.

Library
Workload Manager Library (libwlm.a)

Syntax
#include <sys/wlm.h>

int wlm_key2class ( struct wlm_args *args, wlm_key_t key, void *ctx)

Description
The wlm_key2class subroutine retrieves a class name from a 64-bit key calculated using the
wlm_class2key subroutine. The key-to-class translation is made by going through the WLM configuration
files for the configuration named in the wlm_args structure pointed to by args (or all the WLM
configuration files, if no configuration name is given), and translating all the class names to a 64-bit key
until the matching key is found.

This process is time consuming and WLM offers the subroutines wlm_initkey and wlm_endkey for
applications needing to translate several 64-bit keys back to class names. These subroutines can be used
in conjunction with the wlm_key2class subroutine to speed up searches.

The wlm_initkey subroutine allocates a block of memory, calculates the keys corresponding to the class
names in the configuration(s) in scope, stores the names with the corresponding keys in the memory
buffer, and returns its address. This address is passed to the wlm_key2class subroutine using the ctx
argument, so that wlm_key2class only needs to search through the memory buffer.

After all keys have been translated into class names, the application must call wlm_endkey to free the
memory buffer. Alternatively, for an application translating only one key, it is possible to call
wlm_key2class directly using a null pointer in the ctx argument. This causes the wlm_key2class
subroutine to internally call wlm_initkey and wlm_endkey.

The method of retrieving class names through the WLM configuration files implies that if a class has been
deleted between the time the class name was converted into a key and the call to the wlm_key2class
subroutine, the name corresponding to the key will not be found and the wlm_key2class subroutine
returns an error.

Parameters
- args
  A pointer to a wlm_args structure:
  - confdir field needs to be initialized as described in wlm_initkey if wlm_initkey has not
    been previously invoked (ctx == NULL). Otherwise, the confdir field is ignored.
  - versflags field needs to be initialized with WLM_VERSION and optionally WLM_MUTE.
Return Values
When the `wlm_key2class` operation is successful, the first class name matching the value of the key is returned in the name sub-field of the `wlm_args` structure pointed to by `args`.

Error Codes
If the `wlm_key2class` subroutine is unsuccessful, one of the following error codes is returned:

- `WLM_BADVERS`: Bad version number.
- `WLM_NOT_INITED`: Missing call to `wlm_init`.
- `WLM_NOMEM`: Not enough memory.
- `WLM_NOCLASS`: No class matching the key was found.
- `WLM_EFAULT`: Invalid `ctx` or `args` argument.

Related Information
The `wlm_class2key` subroutine.

The `wlm_endkey` subroutine.

The `wlm_initkey` subroutine.

### wlm_load Subroutine

**Purpose**
Loads a Workload Manager (WLM) configuration into the kernel.

**Library**
Workload Manager Library (`libwlm.a`)

**Syntax**
```c
#include <sys/wlm.h>

int wlm_load (wlmargs);

struct wlm_args *wlmargs;
```

**Description**
The `wlm_load` subroutine loads into the kernel the property files for the WLM configuration passed in the `confdir` field of the `wlmargs` structure. The `confdir` field may also refer to a set of time-based configurations, in which case the appropriate configuration of the set will be loaded and the WLM daemon will later switch to the other configurations of the set on a time basis.

If the WLM is running and `confdir` is not current, this leads to switch to the specified configuration (or configuration set).

If the WLM is running and `confdir` is current, `wlm_load` will refresh the current WLM configuration into the kernel. If a superclass name is given in the `name` field of the class definition substructure, only the subclasses of the given superclass are refreshed. In this context:
- The `wlm_load` subroutine is accessible to root users and to users with administration privileges on the subclasses of the superclass. In all other cases, the `wlm_load` subroutine is only accessible to root users.
- The `wlm_load` subroutine cannot be used to change the mode of operation of WLM (for example, to switch between active and passive modes).
- If `current` is a configuration set, `confdir` must be given in the form `current/config` where `config` is the regular configuration of the set the superclass belongs to. If `config` is the active configuration of the set, the changes will take effect immediately, otherwise they will take effect the next time `config` is made active.

If the caller of `wlm_load` has root privileges and does not specify a superclass, the flags passed in `versflags` can be used to start WLM in active or passive mode, switch between active and passive modes, or enable/disable the rset bindings or the process or class total limits. The `wlm_load` subroutine cannot be used to stop WLM. Use the `wlm_set` subroutine instead.

**Parameter**

`wlmargs` Specifies the address of the `struct wlm_args` data structure containing information about the configuration (or configuration set or superclass) to be loaded and the mode of operation of WLM.

The following fields of the `wlm_args` structure and the embedded substructures can be provided:

- `versflags` Needs to be initialized with WLM_VERSION. May be ORed with WLM_MUTE for `wlm_load` to be silent.
  
  If no change must be done to the mode of operation of WLM, it must be ORed with WLM_TEST_ON (mandatory if superclass is specified).
  
  Otherwise, one of the mutually exclusive flags (WLM_ACTIVE, WLM_CPUONLY, or WLM_PASSIVE) must be given. One or more of the WLM_BIND_RSETS, WLM_PROCTOTAL, or WLM_CLASSTOTAL flags can be given optionally.

- `confdir` Specifies the name of the WLM configuration to be loaded into the kernel. It must be either the name of a valid configuration or configuration set in the `/etc/wlm` subdirectory, the `current` string to refer to the active configuration, or, if superclass is specified and current is a configuration set, it must indicate which configuration of current set the superclass belongs to in the form: `current/config` (this is different from specifying `config` only, which is considered a configuration switch request).

- `name` Specifies the name of a superclass. This is used to refresh only the subclasses of a given superclass.

**Return Values**

Upon successful completion, the `wlm_load` subroutine returns a value of 0. If the `wlm_load` subroutine is unsuccessful, a nonzero value is returned.

**Error Codes**

For a list of the possible error codes returned by the WLM API functions, see the description of the `wlm.h` header file.

**Related Information**

The `wlmcntrl` command.

The `wlm_set` subroutine.
The `wlm.h` header file.

### `wlm_read_classes` Subroutine

#### Purpose
Reads the characteristics of superclasses or subclasses.

#### Library
Workload Manager Library (*libwlm.a*)

#### Syntax

```c
#include <sys/wlm.h>

int wlm_read_classes (wlmargs, class_tbl, nclass);
struct wlm_args *wlmargs;
struct class_definition *class_tbl;
int *nclass;
```

#### Description
The `wlm_read_classes` subroutine is used to get the characteristics of the superclasses or the subclasses of a given subclass of a Workload Manager (WLM) configuration.

- If the name of a configuration is passed in the `confdir` field, the `wlm_read_classes` subroutine reads the property files of the classes of the specified configuration. If `confdir` is set to a null string (`"\0"`), `wlm_read_classes` reads the classes' characteristics from the in-core WLM data structures when WLM is on (and returns an error when WLM is off).

  **Note:** These values may be different from the values in the property files of the configuration pointed to by `/etc/wlm/current`. For instance when a WLM administrator has modified the property files for the configuration pointed to by `/etc/wlm/current` but has not refreshed WLM yet. Another example is if applications dynamically created or modified classes through the API without saving the changes in the `current` configuration property files.

  If your application specifically needs to access the properties of the classes as described in the `/etc/wlm/current` configuration, you must specify `current` as the configuration name in `confdir`.

  If the name of a set of time-based configurations is passed in the `confdir` field, the `wlm_read_classes` subroutine reads the classes of the currently applicable configuration of the set.

- If the name of a valid superclass of the given configuration is passed in the `name` field of the `class_descr` substructure of `wlmargs`, `wlm_read_classes` reads the property files for the subclasses of this superclass. If a null string (`"\0"`) is passed in the `name` field, `wlm_read_classes` reads the property files for the superclasses of the WLM configuration described above.

- When `wlm_read_classes` is successful, the characteristics of the superclasses or subclasses are copied into the array of `class_definition` structures pointed to by `class_tbl`. The integer value pointed to by `nclass` indicates the maximum number of class definitions to be copied. Upon successful return from the function, this value reflects the actual number of classes read.

  If the number of elements copied by `wlm_read_classes` is strictly smaller than the number of elements passed as an argument, all the classes have been read. If it is equal, it may mean that some classes were not copied into the `class_tbl` array because its size is too small.

  The maximum number of classes read by `wlm_read_classes` is 67 (64 user-defined superclasses plus System, Shared and Default) when reading superclasses and 63 (61 user-defined subclasses plus Shared and Default) when reading subclasses characteristics.
Upon successful return from `wlm_read_classes`, the substructure `class` of type `struct class_definition` of the structure pointed to by `wlmargs` contains the default values of various class attributes for the returned set of classes.

This operation does not require any special privileges and is accessible to all users.

**Parameter**

`wlmargs` Specifies the address of a `struct wlm_args` data structure.

The following fields of the `wlm_args` structure and the embedded substructures need to be provided:

- **versflags**
  Needs to be initialized with `WLM_VERSION`.

- **confdir**
  Specifies the name of a WLM configuration. It must be either the name of a valid subdirectory of `/etc/wlm` or a null string (starting with `\0`).

- **name**
  Specifies the name of a superclass existing in the specified configuration or a null string.

All the other fields can be left uninitialized.

`class_tbl` Specifies the address of an array of structures of type `struct class_definition`. Upon successful return from `wlm_read_classes`, this array contains the characteristics of the classes read.

`nclass` Specifies the address of an integer containing the maximum number of element (class definitions) for `wlm_read_classes` to copy into the array above. If the call to `wlm_read_classes` is successful, this integer contains the number of elements actually copied.

**Return Values**

Upon successful completion, the `wlm_read_classes` subroutine returns a value of 0. If the `wlm_read_classes` subroutine is unsuccessful, a nonzero value is returned.

**Error Codes**

For a list of the possible error codes returned by the WLM API functions, see the description of the `wlm.h` header file.

**Related Information**

The `lsclass` command.

The `wlm.h` header file.

**wlm_set Subroutine**

**Purpose**

Sets or queries the Workload Manager (WLM) state.

**Library**

Workload Manager Library (`libwlm.a`)
Syntax

```c
#include <sys/wlm.h>

int wlm_set (flags)
int *flags;
```

Description

The `wlm_set` subroutine is used to set, change, or query the mode of operations of WLM. The state of WLM can be:

- **OFF**
  - Does not classify processes, monitor or regulate resource utilization.
- **ON in passive mode**
  - Classifies the processes and monitors their resource usage but does no regulation.
- **ON in active mode**
  - Specifies the normal operating mode where WLM classifies processes, monitors and regulates the resource usage.

Parameters

- **flags**
  - Specifies the address of an integer interpreted in a manner similar to the `versflags` field of the `wimargs` structure passed to the other API routines. The integer pointed to by `flags` should be initialized with `WLM_VERSION`. In addition, one or more of the following values can be or’ed to `WLM_VERSION`:
    - `WLM_TEST_ON` queries the state of WLM without altering it.
    - `WLM_OFF` turns WLM off.
    - `WLM_ACTIVE` turns WLM on in active mode or transitions from any mode to active mode.
    - `WLM_CPU_ONLY` turns WLM on in active mode for CPU resource only, or transitions from any mode to this mode. This is the same as `WLM_ACTIVE`, but only CPU resources are regulated. Other resources (memory, disk IO, and total limits when enabled) are still accounted.
    - `WLM_PASSIVE` turns WLM on in passive mode or transitions from any mode to passive mode.
    - `WLM_BIND_RSETS` requests that WLM takes the resource set bindings into account.
    - `WLM_PROCTOTAL` enables process total limits on resource usage.
    - `WLM_CLASSTOTAL` enables class total limits on resource usage.

Some combinations of the flags above are not legal:

- `WLM_OFF, WLM_ACTIVE, WLM_CPU_ONLY`, and `WLM_PASSIVE` are mutually exclusive.
- `WLM_BIND_RSETS, WLM_PROCTOTAL, and WLM_CLASSTOTAL` are ineffective when used together with `WLM_OFF`.
- Only `WLM_TEST_ON` is allowed to non-root users.
- If `WLM_TEST_ON` is specified, the other flags are ineffective and should not be specified.

Return Values

Upon successful completion, the `wlm_set` subroutine returns a value of 0, and the current state of WLM is returned in the `flags` parameter. The return value is `WLM_OFF, WLM_ACTIVE, WLM_CPU_ONLY`, or
**WLM_PASSIVE.** When WLM is on in either mode, the **WLM_BIND_RSETS, WLM_PROCTOTAL, and WLM_CLASSTOTAL**, flags are added when appropriate.

**Error Codes**

For a list of the possible error codes returned by the WLM API functions, see the description of the \texttt{wlm.h} header file.

**Related Information**

The \texttt{wlmcntrl} command.

The \texttt{wlm.h} header file.

The \texttt{wlm_load} \texttt{(wlm_load Subroutine on page 542)} subroutine.

---

**wlm_set_tag Subroutine**

**Purpose**

Sets the current process's tag and related flags.

**Library**

Workload Manager Library (\texttt{libwlm.a})

**Syntax**

```c
#include <sys/wlm.h>
#include <sys/user.h>

int wlm_set_tag (tag, flags)
char *tag;
int *flags;
```

**Description**

The \texttt{tag} attribute is a new attribute of a process that can be set using the Workload Manager (WLM) \texttt{wlm_set_tag} subroutine. This tag is a character string with a maximum length of \texttt{WLM_TAG_LENGTH} (not including the null terminator). Process tags can be displayed using the \texttt{ps} command.

The \texttt{tag} attribute is also one of the \texttt{process} attributes used in the assignment rules to automatically assign a process to a given class. The syntax of the assignment rules precludes the use of special characters in the application tag string. Thus, application tags should be comprised only of upper and lower case letters, numbers and underscores ('_').

The main use of the \texttt{tag} attribute is to allow WLM administrators to discriminate between several instances of the same application, which typically have the same user and group ids, execute the same binary, and, therefore, end up in the same class using the standard classification criteria.

For more details about application tags, refer to \texttt{Workload Manager Application Programming Interface}.

When an application sets its tag using \texttt{wlm_set_tag}, it is automatically reclassified according to the current assignment rules and the new tag is taken into account when doing this reclassification.
In addition to the tag itself, the application can also specify flags indicating to WLM if a child process should inherit the tag from its parent after a fork or an exec subroutine.

A process does not require any special privileges to set its tag.

**Parameters**

- **tag**: Specifies the address of a character string. An error is returned if this tag is too long.
- **flags**: Specifies the address of an integer interpreted in a manner similar to the versflags field of the wlmargs structure passed to other API routines. The integer pointed to by flags should be initialized with WLM_VERSION. In addition, one or more of the following values can be or’ed to WLM_VERSION:
  - SWLMTAGINHERITFORK: Specifies that the children of this process inherit the parent’s tag on the fork subroutine.
  - SWLMTAGINHERITEXEC: Specifies that the process retains its tag after a call to the exec subroutine.

Both flags can be set to specify that the children of a tagged process inherits the tag on the fork subroutine and then retains it on the exec subroutine.

**Return Values**

Upon successful completion, the wlm_set_tag subroutine returns a value of 0. In case of error, a non-0 value is returned.

**Error Codes**

For a list of the possible error codes returned by the WLM API functions, see the description of the wlm.h header file.

**Related Information**

The wlm.h header file.

[Workload Manager rules File in AIX 5L Version 5.3 Files Reference.]

---

**wmemchr Subroutine**

**Purpose**

Find a wide-character in memory.

**Library**

Standard library (libc.a)

**Syntax**

```c
#include <wchar.h>

wchar_t *wmemchr (const wchar_t *ws, wchar_t wc, size_t n);
```
Description
The `wmemchr` function locates the first occurrence of `wc` in the initial `n` wide-characters of the object pointed to be `ws`. This function is not affected by locale and all `wchar_t` values are treated identically. The null wide-character and `wchar_t` values not corresponding to valid characters are not treated specially.

If `n` is zero, `ws` must be a valid pointer and the function behaves as if no valid occurrence of `wc` is found.

Return Values
The `wmemchr` function returns a pointer to the located wide-character, or a null pointer if the wide-character does not occur in the object.

Related Information
The `wmemcmp` subroutine, `wmemcpy` subroutine, `wmemmove` subroutine, `wmemset` subroutine.

**wmemcmp Subroutine**

**Purpose**
Compare wide-characters in memory.

**Library**
Standard library (`libc.a`)

**Syntax**
```
#include <wchar.h>
int wmemcmp (const wchar_t * ws1, const wchar_t * ws2, size_t n);
```

**Description**
The `wmemcmp` function compares the first `n` wide-characters of the object pointed to by `ws1` to the first `n` wide-characters of the object pointed to by `ws2`. This function is not affected by locale and all `wchar_t` values are treated identically. The null wide-character and `wchar_t` values not corresponding to valid characters are not treated specially.

If `n` is zero, `ws1` and `ws2` must be a valid pointers and the function behaves as if the two objects compare equal.

**Return Values**
The `wmemcmp` function returns an integer greater than, equal to, or less than zero, accordingly as the object pointed to by `ws1` is greater than, equal to, or less than the object pointed to by `ws2`.

**Related Information**
The `wmemchr` subroutine, `wmemcpy` subroutine, `wmemmove` subroutine, `wmemset` subroutine.
wmemcpy Subroutine

Purpose
Copy wide-characters in memory.

Library
Standard library (libc.a)

Syntax
#include <wchar.h>
wchar_t *wmemcpy (wchar_t * ws1, const wchar_t * ws2, size_t n);

Description
The wmempcy function copies n wide-characters from the object pointed to by ws2 to the object pointed to be ws1. This function is not affected by locale and all wchar_t values are treated identically. The null wide-character and wchar_t values not corresponding to valid characters are not treated specially.

If n is zero, ws1 and ws2 must be a valid pointers, and the function copies zero wide-characters.

Return Values
The wmempcy function returns the value of ws1.

Related Information
The wmemchr subroutine, wmemcmp subroutine, wmemmove subroutine, wmemset subroutine.

wmemmove Subroutine

Purpose
Copy wide-characters in memory with overlapping areas.

Library
Standard library (libc.a)

Syntax
#include <wchar.h>
wchar_t *wmemmove (wchar_t * ws1, const wchar_t * ws2, size_t n);

Description
The wmemmove function copies n wide-characters from the object pointed to by ws2 to the object pointed to by ws1. Copying takes place as if the n wide-characters from the object pointed to by ws2 are first copied into a temporary array of n wide-characters that does not overlap the objects pointed to by ws1 or ws2, and then the n wide-characters from the temporary array are copied into the object pointed to by ws1.

This function is not affected by locale and all wchar_t values are treated identically. The null wide-character and wchar_t values not corresponding to valid characters are not treated specially.
If \( n \) is zero, \( ws1 \) and \( ws2 \) must be a valid pointers, and the function copies zero wide-characters.

**Return Values**

The `wmemmove` function returns the value of \( ws1 \).

**Related Information**

The `wmemchr` subroutine, `wmemcmp` subroutine, `wmemcpy` subroutine, `wmemset` subroutine.

---

**wmemset Subroutine**

**Purpose**

Set wide-characters in memory.

**Library**

Standard library (`libc.a`)

**Syntax**

```c
#include <wchar.h>

wchar_t *wmemset (wchar_t * ws, wchar_t wc, size_t n);
```

**Description**

The `wmemset` function copies the value of \( wc \) into each of the first \( n \) wide-characters of the object pointed to by \( ws \). This function is not affected by locale and all wchar_t values are treated identically. The null wide-character and wchar_t values not corresponding to valid characters are not treated specially. If \( n \) is zero, \( ws \) must be a valid pointer and the function copies zero wide-characters.

**Return Values**

The `wmemset` functions returns the value of \( ws \).

**Related Information**

The `wmemchr` subroutine, `wmemcmp` subroutine, `wmemcpy` subroutine, `wmemmove` subroutine.

---

**wordexp Subroutine**

**Purpose**

Expands tokens from a stream of words.

**Library**

Standard C Library (`libc.a`)

**Syntax**

```c
#include <wordexp.h>
```
int wordexp (const char *Words, wordexp_t *Pwordexp, int Flags);

const char *Words;
wordexp_t *Pwordexp;
int Flags;

Description

The wordexp subroutine performs word expansions equivalent to the word expansion that would be performed by the shell if the contents of the Words parameter were arguments on the command line. The list of expanded words are placed in the Pwordexp parameter. The expansions are the same as that which would be performed by the shell if the Words parameter were the part of a command line representing the parameters to a command. Therefore, the Words parameter cannot contain an unquoted character or any of the unquoted shell special characters | (pipe), & (ampersand), ; (semicolon), < (less than sign), or > (greater than sign), except in the case of command substitution. The Words parameter also cannot contain unquoted parentheses or braces, except in the case of command or variable substitution. If the Words parameter contains an unquoted comment character # (number sign) that is the beginning of a token, the wordexp subroutine may treat the comment character as a regular character, or may interpret it as a comment indicator and ignore the remainder of the expression in the Words parameter.

The wordexp subroutine allows an application to perform all of the shell's expansions on a word or words obtained from a user. For example, if the application prompts for a file name (or a list of file names) and then uses the wordexp subroutine to process the input, the user could respond with anything that would be valid as input to the shell.

The wordexp subroutine stores the number of generated words and a pointer to a list of pointers to words in the Pwordexp parameter. Each individual field created during the field splitting or path name expansion is a separate word in the list specified by the Pwordexp parameter. The first pointer after the last last token in the list is a null pointer. The expansion of special parameters *, @ (at sign), # (number sign), ? (question mark), - (minus sign), $ (dollar sign), ! (exclamation point), and 0 is unspecified.

The words are expanded in the order shown below:
1. Tilde expansion is performed first.
2. Parameter expansion, command substitution, and arithmetic expansion are performed next, from beginning to end.
3. Field splitting is then performed on fields generated by step 2, unless the IFS (input field separators) is full.
4. Path-name expansion is performed, unless the set -f command is in effect.
5. Quote removal is always performed last.

Parameters

<table>
<thead>
<tr>
<th>Flags</th>
<th>Contains a bit flag specifying the configurable aspects of the wordexp subroutine.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pwordexp</td>
<td>Contains a pointer to a wordexp_t structure.</td>
</tr>
<tr>
<td>Words</td>
<td>Specifies the string containing the tokens to be expanded.</td>
</tr>
</tbody>
</table>

The value of the Flags parameter is the bitwise, inclusive OR of the constants below, which are defined in the wordexp.h file.

<table>
<thead>
<tr>
<th>WRDE_APPEND</th>
<th>Appends words generated to those generated by a previous call to the wordexp subroutine.</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRDE_DOOFFS</td>
<td>Makes use of the we_offs structure. If the WRDE_DOOFFS flag is set, the we_offs structure is used to specify the number of null pointers to add to the beginning of the we_words structure. If the WRDE_DOOFFS flag is not set in the first call to the wordexp subroutine with the Pwordexp parameter, it should not be set in subsequent calls to the wordexp subroutine with the Pwordexp parameter.</td>
</tr>
</tbody>
</table>
The **WRDE_APPEND** flag can be used to append a new set of words to those generated by a previous call to the wordexp subroutine. The following rules apply when two or more calls to the wordexp subroutine are made with the same value of the Pwordexp parameter and without intervening calls to the wordfree subroutine:

1. The first such call does not set the **WRDE_APPEND** flag. All subsequent calls set it.
2. For a single invocation of the wordexp subroutine, all calls either set the **WRDE_DOOFFS** flag, or do not set it.
3. After the second and each subsequent call, the Pwordexp parameter points to a list containing the following:
   a. Zero or more null characters, as specified by the **WRDE_DOOFFS** flag and the we_offs structure.
   b. Pointers to the words that were in the Pwordexp parameter 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 the Pwordexp parameter is the total number of words from all of the calls.
5. The application should not modify the Pwordexp parameter between the calls.

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 to a file.

Unless the **WRDE_SHOWERR** flag is set in the Flags parameter, the wordexp subroutine redirects standard error to the /dev/null file for any utilities executed as a result of command substitution while expanding the Words parameter. If the **WRDE_SHOWERR** flag is set, the wordexp subroutine may write messages to standard error if syntax errors are detected while expanding the Words parameter.

The Pwordexp structure is allocated by the caller, but memory to contain the expanded tokens is allocated by the wordexp subroutine and added to the structure as needed.

The Words parameter cannot contain any <newline> characters, or any of the unquoted shell special characters |, & , ; , () , {}, < , or >, except in the context of command substitution.

**Return Values**

If no errors are encountered while expanding the Words parameter, the wordexp subroutine returns a value of 0. If an error occurs, it returns a nonzero value indicating the error.

**Errors**

If the wordexp subroutine terminates due to an error, it returns one of the nonzero constants below, which are defined in the wordexp.h file.

- **WRDE_BADCHAR** One of the unquoted characters |, & , ; , < , >, parenthesis, or braces appears in the Words parameter in an inappropriate context.
- **WRDE_BADVAL** Reference to undefined shell variable when the **WRDE_UNDEF** flag is set in the Flags parameter.
- **WRDE_CMDSUB** Command substitution requested when the **WRDE_NOCMD** flag is set in the Flags parameter.
- **WRDE_NOSPACE** Attempt to allocate memory was unsuccessful.
If the `wordexp` subroutine returns the error value `WRDE_SPACE`, then the expression in the `Pwordexp` parameter is updated to reflect any words that were successfully expanded. In other cases, the `Pwordexp` parameter is not modified.

**Related Information**
The `glob` subroutine, `wordfree` subroutine.

For more information on basic and extended regular expressions, see [Manipulating Strings with sed](#).

---

**wordfree Subroutine**

**Purpose**
Frees all memory associated with the `Pwordexp` parameter.

**Library**
Standard C Library (`libc.a`)

**Syntax**
```c
#include <wordexp.h>

void wordfree (Pwordexp)
    wordexp_t *Pwordexp;
```

**Description**
The `wordfree` subroutine frees any memory associated with the `Pwordexp` parameter from a previous call to the `wordexp` subroutine.

**Parameters**
- `Pwordexp` Structure containing a list of expanded words.

**Related Information**
The `wordexp` subroutine.

---

**write, writex, writev, writexv or pwrite Subroutines**

**Purpose**
Writes to a file.

**Library**
Standard C Library (`libc.a`)

**Syntax**
```c
#include <unistd.h>
```
ssize_t write (FileDescriptor, Buffer, NBytes)
int FileDescriptor;
const void *Buffer;
size_t NBytes;

int writex (FileDescriptor, Buffer, NBytes, Extension)
int FileDescriptor;
char *Buffer;
unsigned int NBytes;
int Extension;
#include <sys/uio.h>

ssize_t writev (FileDescriptor, iov, iovCount)
int FileDescriptor;
const struct iovec *iov;
int iovCount;

ssize_t writevx (FileDescriptor, iov, iovCount, Extension)
int FileDescriptor;
struct iovec *iov;
int iovCount;
int Extension;

ssize_t pwrite (FileDescriptor, Buffer, NBytes, Offset)
int FileDescriptor;
const void *Buffer;
size_t NBytes;
off_t Offset;

Description

The write subroutine attempts to write the number of bytes of data specified by the NBytes parameter to the file associated with the FileDescriptor parameter from the buffer pointed to by the Buffer parameter.

The writev subroutine performs the same action but gathers the output data from the iovCount buffers specified by the array of iovec structures pointed to by the iov parameter. Each iovec entry specifies the base address and length of an area in memory from which data should be written. The writev subroutine always writes a complete area before proceeding to the next.

The writex and writevx subroutines are the same as the write and writev subroutines, respectively, with the addition of an Extension parameter, which is used when writing to some device drivers.

With regular files and devices capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file pointer. Upon return from the write subroutine, the file pointer increments by the number of bytes actually written.

With devices incapable of seeking, writing always takes place starting at the current position. The value of a file pointer associated with such a device is undefined.

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 thread.
Fewer bytes can be written than requested if there is not enough room to satisfy the request. In this case the number of bytes written is returned. The next attempt to write a nonzero number of bytes is unsuccessful (except as noted in the following text). The limit reached can be either that set by the `ulimit` subroutine or the end of the physical medium.

Successful completion of a `write` subroutine clears the SetUserID bit (`S_ISUID`) of a file if all of the following are true:
- The calling process does not have root user authority.
- The effective user ID of the calling process does not match the user ID of the file.
- The file is executable by the group (`S_IXGRP`) or other (`S_IXOTH`).

The `write` subroutine clears the SetGroupID bit (`S_ISGID`) if all of the following are true:
- The calling process does not have root user authority.
- The group ID of the file does not match the effective group ID or one of the supplementary group IDs of the process.
- The file is executable by the owner (`S_IXUSR`) or others (`S_IXOTH`).

**Note:** Clearing of the SetUserID and SetGroupID bits can occur even if the `write` subroutine is unsuccessful, if file data was modified before the error was detected.

If the `O_APPEND` flag of the file status is set, the file offset is set to the end of the file prior to each write.

If the `FileDescriptor` parameter refers to a regular file whose file status flags specify `O_SYNC`, this is a synchronous update (as described in the `open` subroutine).

If the `FileDescriptor` parameter refers to a regular file that a process has opened with the `O_DEFER` file status flag set, the data and file size are not updated on permanent storage until a process issues an `fsync` subroutine or performs a synchronous update. If all processes that have the file open with the `O_DEFER` file status flag set close the file before a process issues an `fsync` subroutine or performs a synchronous update, the data and file size are not updated on permanent storage.

Write requests to a pipe (or first-in-first-out (FIFO)) are handled the same as a regular file with the following exceptions:
- There is no file offset associated with a pipe; hence, each write request appends to the end of the pipe.
- If the size of the write request is less than or equal to the value of the `PIPE_BUF` system variable (described in the `pathconf` routine), the `write` subroutine is guaranteed to be atomic. The data is not interleaved with data from other write processes on the same pipe. Writes of greater than `PIPE_BUF` bytes can have data interleaved, on arbitrary boundaries, with writes by other processes, whether or not the `O_NDELAY` or `O_NONBLOCK` file status flags are set.
- If the `O_NDELAY` and `O_NONBLOCK` file status flags are clear (the default), a write request to a full pipe causes the process to block until enough space becomes available to handle the entire request.
- If the `O_NDELAY` file status flag is set, a write to a full pipe returns a 0.
- If the `O_NONBLOCK` file status flag is set, a write to a full pipe returns a value of -1 and sets the `errno` global variable to `EAGAIN`.

When attempting to write to a character special file that supports nonblocking writes and no data can currently be written (streams are an exception described later in this article):
- If the `O_NDELAY` and `O_NONBLOCK` flags are clear (the default), the `write` subroutine blocks until data can be written.
- If the `O_NDELAY` flag is set, the `write` subroutine returns 0.
- If the `O_NONBLOCK` flag is set, the `write` subroutine returns -1 and sets the `errno` global variable to `EAGAIN` if no data can be written.
When attempting to write to a regular file that supports enforcement-mode record locks, and all or part of the region to be written is currently locked by another process, the following can occur:

- If the `O_NDELAY` and `O_NONBLOCK` file status flags are clear (the default), the calling process blocks until the lock is released.
- If the `O_NDELAY` or `O_NONBLOCK` file status flag is set, then the `write` subroutine returns a value of -1 and sets the `errno` global variable to `EAGAIN`.

**Note:** The `fcntl` subroutine provides more information about record locks.

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:

- `O_NONBLOCK` should specify either `O_NONBLOCK` or `O_NDELAY`. The IBM streams implementation treats these two the same.
- If `O_NONBLOCK` or `O_NDELAY` 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` or `O_NDELAY` is set and the STREAM cannot accept data, `write` will return -1 and set `errno` to `EAGAIN`.
- If `O_NONBLOCK` or `O_NDELAY` 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.

**Note:** The IBM streams implementation treats `O_NONBLOCK` and `O_NDELAY` the same.

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 behavior is unspecified.

If the sum of the `iov_len` values is greater than `SSIZE_MAX`, the operation fails and no data is transferred.

The behavior of an interrupted `write` subroutine depends on how the handler for the arriving signal was installed. The handler can be installed in one of two ways, with the following results:
• If the handler was installed with an indication that subroutines should not be restarted, the `write` subroutine returns a value of -1 and sets the `errno` global variable to `EINTR` (even if some data was already written).

• If the handler was installed with an indication that subroutines should be restarted, and:
  – If no data had been written when the interrupt was handled, the `write` subroutine will not return a value (it is restarted).
  – If data had been written when the interrupt was handled, this `write` subroutine returns the amount of data already written.

**Note:** A write to a regular file is not interruptible. Only writes to objects that may block indefinitely, such as FIFOs, sockets, and some devices, are generally interruptible. If `fildes` refers to a socket, `write` is equivalent to the `send` subroutine with no flags set.

The `pwrite` function performs the same action as `write`, except that it writes into a given position without changing the file pointer. The first three arguments to `pwrite` are the same as `write` with the addition of a fourth argument offset for the desired position inside the file.

**Note:** The `pwrite64` subroutine applies to AIX 4.3 and later.

```c
ssize_t pwrite64(int fd, const void *buf, size_t nbytes, off64_t offset)
```

The `pwrite64` subroutine performs the same action as `pwrite` but the limit of offset to the maximum file size for the file associated with the fileDescriptor and DEV_OFF_MAX if the file associated with fileDescriptor is a block special or character special file.

Using the `write` or `pwrite` subroutine with a file descriptor obtained from a call to the `shm_open` subroutine fails with `ENXIO`.

**Parameters**

- **Buffer**
  Identifies the buffer containing the data to be written.

- **Extension**
  Provides communication with character device drivers that require additional information or return additional status. Each driver interprets the `Extension` parameter in a device-dependent way, either as a value or as a pointer to a communication area. Drivers must apply reasonable defaults when the `Extension` parameter value is 0.

- **FileDescriptor**
  Identifies the object to which the data is to be written.

- **iov**
  Points to an array of `iovec` structures, which identifies the buffers containing the data to be written. The `iovec` structure is defined in the `sys/uio.h` file and contains the following members:

  ```c
  caddr_t iov_base;
  size_t iov_len;
  ```

- **iovCount**
  Specifies the number of `iovec` structures pointed to by the `iov` parameter.

- **NBytes**
  Specifies the number of bytes to write.

**Return Values**

Upon successful completion, the `write`, `writex`, `writev`, and `writevx` subroutines return the number of bytes that were actually written. The number of bytes written is never greater than the value specified by the `NBytes` parameter. Otherwise, a value of -1 is returned and the `errno` global variable is set to indicate the error.

**Error Codes**

The `write`, `writex`, `writev`, and `writevx` subroutines are unsuccessful when one of the following is true:

- **EAGAIN** The `O_NONBLOCK` flag is set on this file and the process would be delayed in the write operation; or an enforcement-mode record lock is outstanding in the portion of the file that is to be written.
The **FileDescriptor** parameter does not specify a valid file descriptor open for writing.

**EDQUOT**
New disk blocks cannot be allocated for the file because the user or group quota of disk blocks has been exhausted on the file system.

**EFBIG**
An offset greater than **MAX_FILESIZE** was requested on the 32-bit kernel.

**EFAULT**
The **Buffer** parameter or part of the **iov** parameter points to a location outside of the allocated address space of the process.

**EFBIG**
An attempt was made to write a file that exceeds the process’ file size limit or the maximum file size. If the user has set the environment variable **XPG_SUS_ENV=ON** prior to execution of the process, then the **SIGXFSZ** signal is posted to the process when exceeding the process’ file size limit.

**EINVAL**
The file position pointer associated with the **FileDescriptor** parameter was negative; the **iovCount** parameter value was not between 1 and 16, inclusive; or one of the **iov_len** values in the **iov** array was negative.

**EINVAL**
The sum of the **iov_len** values from a 32-bit application overflowed a 32-bit signed integer in either a 32-bit or a 64-bit kernel environment, or the sum of the **iov_len** values from a 64-bit application overflowed a 32-bit signed integer in a 32-bit kernel environment.

**EINVAL**
The STREAM or multiplexer referenced by **FileDescriptor** is linked (directly or indirectly) downstream from a multiplexer.

**EINVAL**
The value of the **Nbytes** parameter that is larger than **OFF_MAX**, was requested on the 32-bit kernel. This is a case where the system call is requested from a 64-bit application that is running on a 32-bit kernel.

**EINVAL**
A signal was caught during the write operation, and the signal handler was installed with an indication that subroutines are not to be restarted.

**EIO**
An I/O error occurred while writing to the file system; 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 has no parent process.

**ENOSPC**
No free space is left on the file system containing the file.

**ENXIO**
A hangup occurred on the STREAM being written to.

The **write** or **pwrite** subroutine was used with a file descriptor obtained from a call to the **shm_open** subroutine.

**EPIPE**
An attempt was made to write to a file that is not opened for reading by any process, or to a socket of type **SOCK_STREAM** that is not connected to a peer socket; or an attempt was made to write to a pipe or FIFO that is not open for reading by any process. If this occurs, a **SIGPIPE** signal will also be sent to the process.

**ERANGE**
The transfer request size was outside the range supported by the STREAMS file associated with **FileDescriptor**.

The **write**, **writex**, **writev**, and **writevx** subroutines may be unsuccessful if the following is true:

**ENXIO**
A request was made of a nonexistent device, or the request was outside the capabilities of the device.

**EFBIG**
An attempt was made to write to a regular file where NBytes greater than zero and the starting offset is greater than or equal to the offset maximum established in the open file description associated with **FileDescriptor**.

**EINVAL**
The offset argument is invalid. The value is negative.

**ESPIPE**
**fildes** is associated with a pipe or FIFO.

### Related Information

The **fcntl**, **dup**, or **dup2** subroutine, **fsync** subroutine, **ioctl** subroutine, **lockfx** subroutine, **seek** subroutine, **open**, **openx**, or **creat** subroutine, **pathconf** subroutine, **pipe** subroutine, **poll** subroutine, **select** (*“select Subroutine” on page 139*) subroutine, **ulimit** ("ulimit Subroutine” on page 464) subroutine.

The **limits.h** file, **unistd.h** file.

The **Input and Output Handling Programmer’s Overview** in **AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs**.
wstring Subroutine

Purpose
Perform operations on wide character strings.

Library
Standard C Library (libc.a)

Syntax
#include <wstring.h>

wchar_t *wstrcat ("wstring Subroutine") (XString1, XString2)
wchar_t *XString1, *XString2;

wchar_t * wstrncat (XString, XString2, Number)
wchar_t *XString1, *XString2;
int Number;

int wstrcmp (XString1, XString2)
wchar_t *XString1, *XString2;

int wstrncmp (XString1, XString2, Number)
wchar_t *XString1, *XString2;
int Number;

wchar_t * wstrcpy (XString1, XString2)
wchar_t *XString1, *XString2;

wchar_t * wstrncpy (XString1, XString2, Number)
wchar_t *XString1, *XString2;
int Number;

int wstrlen (XString)
wchar_t *XString;

wchar_t * wstrchr (XString, Number)
wchar_t *XString;
int Number;

wchar_t * wstrchr (XString, Number)
wchar_t *XString;
int Number;

wchar_t * wstrchr (XString1, XString2)
wchar_t *XString1, *XString2;

int wstrspn (XString1, XString2)
wchar_t *XString1, XString2;

int wstrcspn (XString1, XString2)
wchar_t *XString1, XString2;

wchar_t * wstrtok (XString1, XString2)
wchar_t *XString1, XString2;
wchar_t * MyApp\[_\text{wstrdup} (XString1);\]

**Description**

The **wstring** subroutines copy, compare, and append strings in memory, and determine location, size, and existence of strings in memory. For these subroutines, a string is an array of **wchar_t** characters, terminated by a null character. The **wstring** subroutines parallel the **string** subroutines, but operate on strings of type **wchar_t** rather than on type **char**, except as specifically noted below.

The parameters **XString1**, **XString2**, and **XString** point to strings of type **wchar_t** (arrays of wchar characters terminated by a wchar_t null character).

The subroutines **wstrcat**, **wstrncat**, **wstrcpy**, and **wstrncpy** all alter the **XString1** parameter. They do not check for overflow of the array pointed to by **XString1**. All string movement is performed wide character by wide character. Overlapping moves toward the left work as expected, but overlapping moves to the right may give unexpected results. All of these subroutines are declared in the **wstring.h** file.

The **wstrcat** subroutine appends a copy of the **wchar_t** string pointed to by the **XString2** parameter to the end of the **wchar_t** string pointer to by the **XString1** parameter. The **wstrcat** subroutine returns a pointer to the null-terminated result.

The **wstrncat** subroutine copies, at most, the value of the **Number** parameter of **wchar_t** characters in the **XString2** parameter to the end of the **wchar_t** string pointer to by the **XString1** parameter. Copying stops before **Number** **wchar_t** character if a null character is encountered in the string pointer to by the **XString2** parameter. The **wstrncat** subroutine returns a pointer to the null-terminated result.

The **wstrcmp** subroutine lexicographically compares the **wchar_t** string pointed to by the **XString1** parameter to the **wchar_t** string pointed to by the **XString2** parameter. The **wstrcmp** subroutine returns a value that is:

- Less than 0 if **XString1** is less than **XString2**
- Equal to 0 if **XString1** is equal to **XString2**
- Greater than 0 if **XString1** is greater than **XString2**

The **wstrnncmp** subroutine makes the same comparison as **wstrcmp**, but it compares, at most, the value of the **Number** parameter of pairs of **wchar** characters. The comparisons are based on collation values as determined by the locale category **LC_COLLATE** and the **LANG** variable.

The **wstrncpy** subroutine copies the string pointed to by the **XString2** parameter to the array pointed to by the **XString1** parameter. Copying stops when the **wchar_t** null is copied. The **wstrncpy** subroutine returns the value of the **XString1** parameter.

The **wstrncpy** subroutine copies the value of the **Number** parameter of **wchar_t** characters from the string pointed to by the **XString2** parameter to the **wchar_t** array pointed to by the **XString1** parameter. If **XString2** is less than **Number** **wchar_t** characters long, then **wstrncpy** pads **XString1** with trailing null characters to fill **Number** **wchar_t** characters. If **XString2** is **Number** or more **wchar_t** characters long, only the first **Number** **wchar_t** characters are copied; the result is not terminated with a null character. The **wstrncpy** subroutine returns the value of the **XString1** parameter.

The **wstrlen** subroutine returns the number of **wchar_t** characters in the string pointed to by the **XString** parameter, not including the terminating **wchar_t** null.
The wstrchr subroutine returns a pointer to the first occurrence of the wchar_t specified by the Number parameter in the wchar_t string pointed to by the XString parameter. A null pointer is returned if the wchar_t does not occur in the wchar_t string. The wchar_t null that terminates a string is considered to be part of the wchar_t string.

The wstrchr subroutine returns a pointer to the last occurrence of the character specified by the Number parameter in the wchar_t string pointed to by the XString parameter. A null pointer is returned if the wchar_t does not occur in the wchar_t string. The wchar_t null that terminates a string is considered to be part of the wchar_t string.

The wstrpbrk subroutine returns a pointer to the first occurrence in the wchar_t string pointed to by the XString1 parameter of any code point from the string pointed to by the XString2 parameter. A null pointer is returned if no character matches.

The wstrspn subroutine returns the length of the initial segment of the string pointed to by the XString1 parameter that consists entirely of code points from the wchar_t string pointed to by the XString2 parameter.

The wstrcspn subroutine returns the length of the initial segment of the wchar_t string pointed to by the XString1 parameter that consists entirely of code points not from the wchar_t string pointed to by the XString2 parameter.

The wstrtok subroutine returns a pointer to an occurrence of a text token in the string pointed to by the XString1 parameter. The XString2 parameter specifies a set of code points as token delimiters. If the XString1 parameter is anything other than null, then the wstrtok subroutine reads the string pointed to by the XString1 parameter until it finds one of the delimiter code points specified by the XString2 parameter. It then stores a wchar_t null into the wchar_t string, replacing the delimiter code point, and returns a pointer to the first wchar_t of the text token. The wstrtok subroutine keeps track of its position in the wchar_t string so that subsequent calls with a null XString1 parameter step through the wchar_t string. The delimiters specified by the XString2 parameter can be changed for subsequent calls to wstrtok. When no tokens remain in the wchar_t string pointed to by the XString1 parameter, the wstrtok subroutine returns a null pointer.

The wstrdup subroutine returns a pointer to a wchar_t string that is a duplicate of the wchar_t string to which the XString1 parameter points. Space for the new string is allocated using the malloc subroutine. When a new string cannot be created, a null pointer is returned.

Related Information

The malloc subroutine, strcat, strncat, strxfrm, strcpy, strncpy, or strdup subroutine, strcmp, strncmp, strcasecmp, strncasecmp, strncompare, or strcoll subroutine, strlen, strnlen, strnchr, strchr, strpbrk, strspn, strcspn, strstr, or strtok subroutine.

List of String Manipulation Services in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.


Subroutines, Example Programs, and Libraries in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
wstrtod or watof Subroutine

Purpose
Converts a string to a double-precision floating-point.

Library
Standard C Library

Syntax
```
#include <wstring.h>

double wstrtod (String, Pointer)
wchar_t *String, **Pointer;

double watof (String)
wchar_t *String;
```

Description
The wstrtod subroutine returns a double-precision floating-point number that is converted from an wchar_t string pointed to by the String parameter. The system searches the String until it finds the first unrecognized character.

The wstrtod subroutine recognizes a string that starts with any number of white-space characters (defined by the iswspace subroutine), followed by an optional sign, a string of decimal digits that may include a decimal point, e or E, an optional sign or space, and an integer.

When the value of Pointer is not (wchar_t **) null, a pointer to the search terminating character is returned to the address indicated by Pointer. When the resulting number cannot be created, *Pointer is set to String and 0 (zero) is returned.

The watof (String) subroutine functions like the wstrtod (String (wchar_t **) null).

Parameters
- **String** Specifies the address of the string to scan.
- **Pointer** Specifies the address at which the pointer to the terminating character is stored.

Error Codes
When the value causes overflow, HUGE_VAL (defined in the math.h file) is returned with the appropriate sign, and the errno global variable is set to ERANGE. When the value causes underflow, 0 is returned and the errno global variable is set to ERANGE.

Related Information
The atof, atoff, strtod, strtof, scanf, fscanf, sscanf, sscanf, scanf, fscanf, sscanf, or wsscanf subroutine, strtok, strtof, atof, atoi subroutine, wstrtol, watol, watoi subroutine.

Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
wstrtol, watol, or watoi Subroutine

Purpose
Converts a string to an integer.

Library
Standard C Library (libc.a)

Syntax
#include <wstring.h>

long wstrtol (wchar_t *String, **Pointer, int Base);
wxchar_t *String, **Pointer;
int Base;

long watol (wchar_t *String);
int watoi (wchar_t *String);

Description
The wstrtol subroutine returns a long integer that is converted from the string pointed to by the String parameter. The string is searched until a character is found that is inconsistent with Base. Leading white-space characters defined by the ctype subroutine iswspace are ignored.

When the value of Pointer is not (wchar_t **) null, a pointer to the terminating character is returned to the address indicated by Pointer. When an integer cannot be created, the address indicated by Pointer is set to String, and 0 is returned.

When the value of Base is positive and not greater than 36, that value is used as the base during conversion. Leading zeros that follow an optional leading sign are ignored. When the value of Base is 16, 0x and 0X are ignored.

When the value of Base is 0, the system chooses an appropriate base after examining the actual string. An optional sign followed by a leading zero signifies octal, and a leading 0x or 0X signifies hexadecimal. In all other cases, the subroutines assume a decimal base.

Truncation from long data type to int data type occurs by assignment, and also by explicit casting.

The watol (wchar_t *String) subroutine functions like wstrtol (wchar_t *String, **null, 10).

The watoi (wchar_t *String) subroutine functions like (int) wstrtol (wchar_t *String, **null, 10).

Note: Even if overflow occurs, it is ignored.

Parameters
String Specifies the address of the string to scan.
Pointer Specifies the address at which the pointer to the terminating character is stored.
Base Specifies an integer value used as the base during conversion.
Related Information

Subroutines Overview in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

xcrypt_key_setup, xcrypt_encrypt, xcrypt_decrypt, xcrypt_hash,
xcrypt_malloc, xcrypt_free, xcrypt_printb, xcrypt_mac, xcrypt_hmac,
xcrypt_sign, xcrypt_verify, xcrypt_dh_keygen, xcrypt_dh, xcrypt_btoa
and xcrypt_randbuff Subroutine

Purpose

Provides various block and stream cipher algorithms and two crypto-secure hash algorithms.

Library

Cryptographic Library (libmodcrypt.a)

Syntax

#include <xcrypt.h>

int xcrypt_key_setup (alg, key, keymat, keysze, dir)
int alg;
xcrypt_key *key;
uchar *keymat;
int keysize;
int dir;

int xcrypt_encrypt (alg, mode, key, IV, in, insize, out, padding)
int alg;
int mode;
xcrypt_key *key;
uchar *IV;
uchar *in;
int insize;
uchar *out;
int padding;

int xcrypt_decrypt (alg, mode, key, IV, in, insize, out, padding)
int alg;
int mode;
xcrypt_key *key;
uchar *IV;
uchar *in;
int insize;
uchar *out;
int padding;

int xcrypt_hash (alg, in, insize, out)
int alg;
uchar *in;
int insize;
uchar *out;

int xcrypt_malloc (pp, size, blocksize)
uchar **pp;
int size;
int blocksize;
Description
These subroutines provide block and stream cipher algorithms, plus two crypto-secure hash algorithms.
Encryption may be done through the Rijndael, Mars, and Twofish block ciphers or the SEAL stream cipher.
Each of these algorithms uses a block length of 128 bits and key lengths of 128, 192 and 256 bits.
SEAL is a stream cipher that uses a 160 bit key and a 32 bit word input stream. In addition, the MD5 and SHA-1 cryptographic hash algorithms are included.

The libmodcrypt.a library and associated headers are available through the Expansion Pack.

The xcrypt_key_setup subroutine is used to setup a key schedule for any of the block cipher algorithms. It stores the key schedule in the xcrypt_key data structure that is passed in. If key scheduling is done for HMAC, set the dir parameter of xcrypt_key_setup to NULL. Note that when using the Twofish method, the keymat parameter should also be set to NULL. The xcrypt_key_setup subroutine expects the keymat pointer to point to a PKCS#8 object when used with public key encryption. Then the keysize parameter indicates the size of the PKCS#8 object, not the size of the key.
The `xcrypt_encrypt` subroutine encrypts a buffer. Data can be encrypted using the CBC mode (Cipher Block Chaining), EBC mode (Electronic Codebook) or CBF1 mode. Note that when EBC mode is being used, no initialization vector is required.

The `xcrypt_decrypt` subroutine decrypts a buffer. Data can be encrypted using the CBC mode (Cipher Block Chaining), EBC mode (Electronic Codebook) or CBF1 mode. If the `xcrypt_encrypt` subroutine is called with padding on, the `xcrypt_decrypt` subroutine must also be called with padding on. It is the caller’s responsibility to determine whether padding is used. Note that when EBC mode is being used, no initialization vector is required.

The `xcrypt_hash` subroutine hashes a buffer using either the MD5 or SHA-1 algorithm.

The `xcrypt_malloc` subroutine dynamically allocates the least size bytes of memory to provide blocks of `blocksize` bytes. For example, if `size` is 105 and `blocksize` is 10, the `xcrypt_malloc` subroutine will return at least 110 bytes of memory (11 blocks, each 10 bytes in size). The `xcrypt_malloc` subroutine should be used when you need `xcrypt` to pad buffers. It will make sure that enough memory is allocated for the data to be encrypted, plus the padding.

The `xcrypt_free` subroutine overwrites and frees dynamically allocated memory.

The `xcrypt_printb` subroutine prints a buffer to the screen in hexadecimal notation.

The `xcrypt_mac` subroutine provides the caller with a Message Authentication Code (MAC). DES is the only algorithm that is supported.

The `xcrypt_hmac` subroutine provides the caller with a Hashed Message Authentication Code (HMAC). The algorithm used is MD5 or SHA-1.

The `xcrypt_sign` subroutine allows the caller to sign data using public key mechanisms.

The `xcrypt_verify` subroutine allows the caller to verify private key signatures.

The `xcrypt_dh_keygen` subroutine returns the private key object to be used in Diffie-Helman key agreement. The `dh` parameter should point to NULL. The `keysize` parameter can be `KEY_512`, `KEY_1024`, or `KEY_2048`.

The `xcrypt_dh` subroutine allows the caller to execute the two steps to compute a shared secret through the Diffie-Hellman key agreement algorithm. If `in` is NULL, then `out` contains the public shared object to be transmitted to the other party involved in the key agreement. Otherwise, `in` points to the public shared object received from another party, and `out` points to the shared private key.

The `xcrypt_btoa` subroutine returns a string representing the buffer in hexadecimal. Note that the `dest` parameter must point to a buffer of `size * 2 + 1`.

The `xcrypt_randbuff` subroutine fills a buffer with random data.
Parameters

**alg**

Specifies the cipher to use. Use the symbolic constants that are described below:

- **RIJNDAEL**
  Rijndael (AES) block cipher. Supports key sizes of 128, 192, and 256 bits.
- **MARS**
  Mars block cipher. Supports key sizes of 128, 192, and 256 bits.
- **TWOFISH**
  Twofish block cipher. Supports key sizes of 128, 192, and 256 bits.
- **SEAL**
  SEAL stream cipher. Supports key sizes of 128, 192, and 256 bits.
- **SHA1**
  SHA-1 one-way hash function. Arbitrary lengths are permitted.
- **MD5**
  MD5 one-way hash function. Arbitrary lengths are permitted.
- **DES**
  Data Encryption Standard. Supports key sizes of 64 bits.
- **TDES**
  Triple Data Encryption Standard. Supports key sizes of 64 and 128 bits.
- **MAC_DES**
  Message Authentication Code using the DES algorithm. Supports key sizes of 64 bits.
- **CAST5**
  CAST encryption algorithm. Supports key sizes of 40, 80, and 128 bits.
- **RSA**
  Rivest, Shamir Adleman. The *keysize* passed to `xcrypt_key_setup` should be the size of the PKCS#8 object.
- **DSA**
  Digital Signature Algorithm. The *keysize* passed to `xcrypt_key_setup` should be the size of the PKCS#8 object.

**key**

Points to the key instance to set up. Use for encryption or decryption.

**keymat**

Points to the key material used to build the key schedule.

**keysize**

Size of the *keymat* parameter. Use the symbolic constants described below:

- **KEY_64**
  64 bit key
- **KEY_80**
  80 bit key
- **KEY_128**
  128 bit key
- **KEY_192**
  192 bit key
- **KEY_256**
  256 bit key

**dir**

The direction (encryption or decryption). Use the symbolic constants described below:

- **DIR_ENCRYPT**
  Encrypt
- **DIR_DECRYPT**
  Decrypt

**mode**

Specifies the mode of operation. Use the symbolic constants described below:

- **MODE_ECB**
  Ciphering in ECB mode
- **MODE_CBC**
  Ciphering in CBC mode
- **MODE_CFB1**
  Ciphering in 1-bit CFB mode

**IV**

Points to the buffer holding the initialization vector.

**Note:** When using ECB mode, the *IV* parameter should point to NULL.
in Points to the buffer holding the data to encrypt, decrypt, or hash.
insize Contains the size of the in parameter.
mac Points to an output buffer that will hold the MAC.
out Points to a preallocated output buffer.
padding Specifies whether xcrypt should pad the buffers or not. Use the symbolic constants described below:

   TRUE   True
   FALSE  False

pp A double pointer to the destination.
sig Points to an output buffer that holds the RSA signature.
sigsiz The size of sig in bytes.
size Contains the amount of memory to allocate, deallocate, print the contents of, or convert to a string.
blocksize Contains the size of the blocks. Use the symbolic constants described below:

   BITS_32
       32 bits
   BITS_60
       80 bits
   BITS_128
       128 bits
   BITS_160
       160 bits
   BITS_192
       192 bits
   BITS_256
       256 bits
   DES_BLOCKSIZE
       64 bits

p Points to the memory to overwrite and free.
buff Points to a buffer to print or convert to a string.
dest Points to a preallocated destination buffer.
dh_pk Refers to the private key object to be passed to xcrypt_dh. The private key object is obtained by calling xcrypt_dh_keygen before calling xcrypt.

Return Values
The xcrypt_key_setup, xcrypt_hash and xcrypt_dh_keygen subroutines return 0 on success. The xcrypt_malloc subroutine returns the amount of memory allocated on success. The xcrypt_encrypt subroutine returns the amount of data encrypted on success. The xcrypt_decrypt subroutine returns the amount of data decrypted on success.

Upon success, the xcrypt_mac subroutine returns the size of mac in bytes; the xcrypt_hmac subroutine returns the size of hashed mac in bytes; the xcrypt_sig subroutine returns the size of signature; and the xcrypt_dh subroutine returns the number of bytes written to out. The xcrypt_verify subroutine returns a value of 1 to indicate successful signal verification.

On failure the above subroutines return the following error codes:

Error Codes
xcrypt_key_setup:
BAD_ALIGN32 A parameter is not aligned on a 32 bit boundary.
BAD_KEY_DIR
BAD_KEY_INSTANCE
BAD_KEY_MAT

The dir parameter is not valid
The key parameter is not valid
The keysize parameter is not valid or the key parameter is corrupt.

xcrypt_encrypt:

BAD_ALG
BAD_CIPHER_MODE
BAD_CIPHER_STATE
BAD_INPUT_LEN
BAD_IV
BAD_IV_MAT
BAD_KEY_INSTANCE

The alg parameter is not valid.
The mode parameter is not valid.
The key parameter is not valid.
The insize parameter is not a multiple of the blocksize being used by a block cipher for encryption or decryption.
The IV parameter is set to NULL when the mode parameter is set to MODE_CBC.
The IV parameter is not valid.
The key parameter is not valid.

xcrypt_decrypt:

BAD_ALG
BAD_CIPHER_MODE
BAD_CIPHER_STATE
BAD_INPUT_LEN
BAD_IV
BAD_IV_MAT
BAD_KEY_INSTANCE

The alg parameter is not valid.
The mode parameter is not valid.
The key parameter is not valid.
The insize parameter is not a multiple of the blocksize being used by a block cipher for encryption or decryption.
The IV parameter is set to NULL when the mode parameter is set to MODE_CBC.
The IV parameter is not valid.
The key parameter is not valid.

xcrypt_hash:

BAD_ALG

The alg parameter is not valid.

xcrypt_malloc:

BAD_MEM_ALLOC

The system could not allocate size bytes.

yield Subroutine

Purpose
Yields the processor to processes with higher priorities.

Library
Standard C library (libc.a)

Syntax
void yield (void);
Description
The **yield** subroutine forces the current running process or thread to relinquish use of the processor. If the run queue is empty when the **yield** subroutine is called, the calling process or kernel thread is immediately rescheduled. If the calling process has multiple threads, only the calling thread is affected. The process or thread resumes execution after all threads of equal or greater priority are scheduled to run.

Related Information
The **getpriority**, **setpriority**, or **nice** subroutine, **setpri** subroutine.

"setpri Subroutine" on page 183
addch, mvaddch, mvwaddch, or waddch Subroutine

Purpose
Adds a single-byte character and rendition to a window and advances the cursor.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
int addch(const ctype ch);
int mvaddch(int y, int x, const ctype ch);
int mvwaddch(Window *in, const ctype ch);
int waddch(Window *win, const ctype ch);

Description
The addch, waddch, mvaddch, and mvwaddch subroutines add a character to a window at the logical cursor location. After adding the character, curses advances the position of the cursor one character. At the right margin, an automatic new line is performed.

The addch subroutine adds the character to the stdscr at the current logical cursor location. To add a character to a user-defined window, use the waddch and mvwaddch subroutines. The mvaddch and mvwaddch subroutines move the logical cursor before adding a character.

If you add a character to the bottom of a scrolling region, curses automatically scrolls the region up one line from the bottom of the scrolling region if scrollok is enabled. If the character to add is a tab, new-line, or backspace character, curses moves the cursor appropriately in the window to reflect the addition. Tabs are set at every eighth column. If the character is a new-line, curses first uses the wclrtoeol subroutine to erase the current line from the logical cursor position to the end of the line before moving the cursor.

You can also use the addch subroutines to add control characters to a window. Control characters are drawn in the ^X notation.

Adding Video Attributes and Text
Because the Char parameter is an integer, not a character, you can combine video attributes with a character by ORing them into the parameter. The video attributes are also set. With this capability you can copy text and video attributes from one location to another using the inch, mvinch, mvwinch, or winch Subroutine and addch subroutines.

Parameters
ch
y
x
*win
Return Values
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

Examples
1. To add the character H represented by variable x to stdscr at the current cursor location, enter:
   ```c
   chtype x;
   x='H';
   addch(x);
   ```
2. To add the x character to stdscr at the coordinates y = 10, x = 5, enter:
   ```c
   mvaddch(10, 5, 'x');
   ```
3. To add the x character to the user-defined window my_window at the coordinates y = 10, x = 5, enter:
   ```c
   WINDOW *my_window;
   mvwaddch(my_window, 10, 5, 'x');
   ```
4. To add the x character to the user-defined window my_window at the current cursor location, enter:
   ```c
   WINDOW *my_window;
   waddch(my_window, 'x');
   ```
5. To add the character x in standout mode, enter:
   ```c
   waddch(my_window, 'x' | A_STANDOUT);
   ```
   This allows 'x' to be highlighted, but leaves the rest of the window alone.

Related Information
The inch, winch, mvinch, or mvwinch, or winch Subroutine" on page 624 subroutine, wclrtoeol or wcrtom Subroutine.

addnstr, addstr, mvaddnstr, mvaddstr, mvwaddnstr, mvwaddstr, waddnstr, or waddstr Subroutine

Purpose
Adds a string of multi-byte characters without rendition to a window and advances the cursor.

Library
Curses Library (libcurses.a)

Syntax
```c
#include <curses.h>
int addnstr(const char *str,
            int n);
int addstr(const char *str);
int mvaddnstr(int y,
              int x,
              const char *str,
              int n);
int mvaddstr(int y,
              int x,
              const char *str);
```
int mvwaddnstr(WINDOW *win,
int y,
int x,
const char *str,
int n);

int mvwaddstr(WINDOW *win,
int y,
int x,
const char *str);

int waddnstr(WINDOW *win,
const char *str);

int waddstr(WINDOW *win,
const char *str);

Description
These subroutines write the characters of the string str on the current or specified window starting at the current or specified position using the background rendition.

These subroutines advance the cursor position, perform special character processing, and perform wrapping.

The addstr, mvaddstr, mvwaddstr and waddstr subroutines are similar to calling mbstowcs on str, and then calling addwstr, mvaddwstr, mvwaddwstr, and waddwstr, respectively.

The addnstr, mvaddnstr, mvwaddnstr and waddnstr subroutines use at most, n bytes from str. These subroutines add the entire string when n is -1.

Parameters

Column Specifies the horizontal position to move the cursor to before adding the string.
Line Specifies the vertical position to move the cursor to before adding the string.
String Specifies the string to add.
Window Specifies the window to add the string to.

Return Values
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

Examples
1. To add the string represented by xyz to the stdscr at the current cursor location, enter:
   ```c
   char *xyz;
   xyz="Hello!";
   addstr(xyz);
   ```
2. To add the “Hit a Key” string to the stdscr at the coordinates y=10, x=5, enter:
   ```c
   mvaddstr(10, 5, "Hit a Key");
   ```
3. To add the xyz string to the user-defined window my_window at the coordinates y=10, x=5, enter:
   ```c
   mvwaddstr(my_window, 10, 5, "xyz");
   ```
4. To add the xyz string to the user-defined string at the current cursor location, enter:
   ```c
   waddstr(my_window, "xyz");
   ```

Related Information
The addch subroutine.
attroff, attron, attrset, wattroff, wattron, or wattrset Subroutine

**Purpose**
Restricted window attribute control functions.

**Library**
Curses Library (**libcurses.a**)

**Syntax**

```c
#include <curses.h>

int attroff (int *attrs);

int attron (int *attrs);

int attrset (int *attrs);

int wattroff (WINDOW *win, int *attrsr);

int wattron (WINDOW *win, int *attrs);

int wattrset (WINDOW *win, int *attrsr);
```

**Description**
These subroutines manipulate the window attributes of the current or specified window.

The `attroff` and `wattroff` subroutines turn off `attrs` in the current or specified specified window without affecting any others.

The `attron` and `wattron` subroutines turn on `attrs` in the current or specified specified window without affecting any others.
The `attrset` and `wattrset` subroutines set the background attributes of the current or specified specified window to `attrs`.

It unspecified whether these subroutines can be used to manipulate attributes than A_BLINK, A_BOLD, A_DIM, A_REVERSE, A_STANDOUT and A_UNDERLINE.

**Parameters**

* `attrs` Specifies which attributes to turn off.
* `win` Specifies the window in which to turn off the specified attributes.

**Return Values**

These subroutines always return either OK or 1.

**Examples**

For the `attroff` or `wattroff` subroutines:

1. To turn the off underlining attribute in `stdscr`, enter:
   ```c
   attroff(A_UNDERLINE);
   ```
2. To turn off the underlining attribute in the user-defined window `my_window`, enter:
   ```c
   wattroff(my_window, A_UNDERLINE);
   ```

For the `attron` or `wattron` subroutines:

1. To turn on the underlining attribute in `stdscr`, enter:
   ```c
   attron(A_UNDERLINE);
   ```
2. To turn on the underlining attribute in the user-defined window `my_window`, enter:
   ```c
   wattron(my_window, A_UNDERLINE);
   ```

For the `attrset` or `wattrset` subroutines:

1. To set the current attribute in the `stdscr` global variable to blink, enter:
   ```c
   attrset(A_BLINK);
   ```
2. To set the current attribute in the user-defined window `my_window` to blinking, enter:
   ```c
   wattrset(my_window, A_BLINK);
   ```
3. To turn off all attributes in the `stdscr` global variable, enter:
   ```c
   attrset(0);
   ```
4. To turn off all attributes in the user-defined window `my_window`, enter:
   ```c
   wattrset(my_window, 0);
   ```

**Related Information**

The `standend` subroutine.

[Curses Overview for Programming](#) in *AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs*.

[List of Curses Subroutines](#) in *AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs*.

[Setting Video Attributes and Curses Options](#) in *AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs*.
attron or wattron Subroutine

Purpose
Turns on specified attributes.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

attron(Attributes)
cchar *Attributes;

wattron(Window, Attributes)
WINDOW *Window;
cchar *Attributes;

Description
The attron and wattron subroutines turn on specified attributes without affecting any others. The attron subroutine turns the specified attributes on in stdscr. The wattron subroutine turns the specified attributes on in the specified window.

Parameters
Attributes Specifies which attributes to turn on.
Window Specifies the window in which to turn on the specified attributes.

Examples
1. To turn on the underlining attribute in stdscr, enter:
   attron(A_UNDERLINE);
2. To turn on the underlining attribute in the user-defined window my_window, enter:
   wattron(my_window, A_UNDERLINE);

Related Information
Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Setting Video Attributes and Curses Options in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

attrset or wattrset Subroutine

Purpose
Sets the current attributes of a window to the specified attributes.
Libraries
Curses Library (libcurses.a)

Syntax
#include <curses.h>

attrset(Attributes)
cchar *Attributes;

wattrset(Window, Attributes)
WINDOW *Window;
cchar *Attributes;

Description
The attrset and wattrset subroutines set the current attributes of a window to the specified attributes. The attrset subroutine sets the current attribute of stdscr. The wattrset subroutine sets the current attribute of the specified window.

Parameters

Attributes Specifies which attributes to set.
Window Specifies the window in which to set the attributes.

Examples
1. To set the current attribute in the stdscr global variable to blink, enter:
   attrset(A_BLINK);
2. To set the current attribute in the user-defined window my_window to blinking, enter:
   wattrset(my_window, A_BLINK);
3. To turn off all attributes in the stdscr global variable, enter:
   attrset(0);
4. To turn off all attributes in the user-defined window my_window, enter:
   wattrset(my_window, 0);

Related Information
Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Setting Video Attributes and Curses Options in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

baudrate Subroutine

Purpose
Gets the terminal baud rate.

Library
Curses Library (libcurses.a)
Syntax
#include <curses.h>

int baudrate(void)

Description
The **baudrate** subroutine extracts the output speed of the terminal in bits per second.

Return Values
The **baudrate** subroutine returns the output speed of the terminal.

Examples
To query the baud rate and place the value in the user-defined integer variable BaudRate, enter:
BaudRate = baudrate();

Related Information
The **tcgetattr** subroutine.

---

**beep Subroutine**

Purpose
Sounds the audible alarm on the terminal.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

int beep(void);

Description
The **beep** subroutine alerts the user. It sounds the audible alarm on the terminal, or if that is not possible, it flashes the screen (visible bell). If neither signal is possible, nothing happens.

Return Values
The **beep** subroutine always returns OK.

Examples
To sound an audible alarm, enter:
beep();

Related Information
The **flash** subroutine.
box Subroutine

Purpose
Draws borders from single-byte characters and renditions.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

int box(WINDOW *win,
cttype verch,
cttype horch);

Description
The box subroutine draws a border around the edges of the specified window. This subroutine does not advance the cursor position. This subroutine does not perform special character processing or perform wrapping.

The box subroutine (*win, verch, horch) has an effect equivalent to:

wborder(win, verch, verch, horch, horch, 0, 0, 0, 0);

Parameters
horch Specifies the character to draw the horizontal lines of the box. The character must be a 1-column character.
verch Specifies the character to draw the vertical lines of the box. The character must be a 1-column character.
*win Specifies the window to draw the box in or around.

Return Values
Upon successful completion, the box function returns OK. Otherwise, it returns ERR.

Examples
1. To draw a box around the user-defined window, my_window, using | (pipe) as the vertical character and - (minus sign) as the horizontal character, enter:

   WINDOW *my_window;
   box(my_window, '|', '-');

2. To draw a box around my_window using the default characters ACS_VLINE and ACS_HLINE, enter:

   WINDOW *my_window;
   box(my_window, 0, 0);
Related Information

Curses Overview for Programming, List of Curses Subroutines, and Windows® in the Curses Environment in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

can_change_color, color_content, has_colors, init_color, init_pair, start_color or pair_content Subroutine

Purpose

Color manipulation functions and external variables for color support.

Library

Curses Library (libcurses.a)

Syntax

```c
#include <curses.h>

bool can_change_color(void);

int color_content(short color,
                 short *red,
                 short *green,
                 short *blue);

int COLOR_PAIR(int n);

bool has_colors(void);

int init_color (short color,
               short red,
               short green,
               short blue);

int init_pair (short pair, short f, short b);

int pair_content (short pair, short *f, short *b);

int PAIR_NUMBER (int value);
int start_color (void);

extern int COLOR_PAIRS;
extern int COLORS;
```

Description

These functions manipulate color on terminals that support color.

Querying Capabilities

The has_colors subroutine indicates whether the terminal is a color terminal. The can_change_color subroutine indicates whether the terminal is a color terminal on which colors can be redefined.
Initialisation
The start_color subroutine must be called in order to enable use of colors and before any color manipulation function is called. This subroutine initializes eight basic colors (black, blue, green, cyan, red, magenta, yellow, and white) that can be specified by the color macros (such as COLOR_BLACK) defined in <curses.h>. The initial appearance of these eight colors is not specified.

The function also initialises two global external variables:
- COLORS defines the number of colors that the terminal supports. If COLORS is 0, the terminal does not support redefinition of colors (and can_change_color subroutine will return FALSE).
- COLOR_PAIRS defines the maximum number of color-pairs that the terminal supports.

Color Identification
The init_color subroutine redefines color number color, on terminals that support the redefinition of colors, to have the red, green, and blue intensity components specified by red, green, and blue, respectively. Calling init_color subroutine also changes all occurrences of the specified color on the screen to the new definition.

The color_content subroutine identifies the intensity components of color number color. It stores the red, green, and blue intensity components of this color in the addresses pointed to by red, green, and blue, respectively.

For both functions, the color argument must be in the range from 0 to and including COLORS -1. Valid intensity values range from 0 (no intensity component) up to and including 1000 (maximum intensity in that component).

User-Defined Color Pairs
Calling init_pair defines or redefines color-pair number pair to have foreground color f and background color b. Calling init_pair changes any characters that were displayed in the color pair's old definition to the new definition and refreshes the screen.

After defining the color pair, the macro COLOR_PAIR(n) returns the value of color pair n. This value is the color attribute as it would be extracted from a chtype. Conversely, the macro PAIR_NUMBER(value) returns the color pair number associated with the color attribute value.

The pair_content subroutine retrieves the component colors of a color-pair number pair. It stores the foreground and background color numbers in the variables pointed to by f and b, respectively.

With init_pair and pair_content subroutines, the value of pair must be in a range from 0 to and including COLOR_PAIRS -1. (There may be an implementation-specific upper limit on the valid value of pair, but any such limit is at least 63.) Valid values for f and b are the range from 0 to and including COLORS -1.

The can_change_color subroutine returns TRUE if the terminal supports colors and can change their definitions; otherwise, it returns FALSE.

Parameters

```c
void start_color();
void init_color(int color, int red, int green, int blue);
void init_pair(int pair, int f, int b);
```
Return Values
The has_colors subroutine returns TRUE if the terminal can manipulate colors; otherwise, it returns FALSE.

Upon successful completion, the other functions return OK. Otherwise, they return ERR.

Examples
For the can_change_color subroutine:

To test whether or not a terminal can change its colors, enter the following and check the return for TRUE or FALSE:

can_change_color();

For the color_content subroutine:

To obtain the RGB component information for color 10 (assuming the terminal supports at least 11 colors), use:

short *r, *g, *b;
color_content(10,r,g,b);

For the has_color subroutine:

To determine whether or not a terminal supports color, use:

has_colors();

For the pair_content subroutine:

To obtain the foreground and background colors for color-pair 5, use:

short *f, *b;
pair_content(5,f,b);

For this subroutine to succeed, you must have already initialized the color pair. The foreground and background colors will be stored at the locations pointed to by f and b.

For the start_color subroutine:

To enable the color support for a terminal that supports color, use:

start_color();

For the init_pair subroutine:

To initialize the color definition for color-pair 2 to a black foreground (color 0) with a cyan background (color 3), use:

init_pair(2, COLOR_BLACK, COLOR_CYAN);

For the init_color subroutine:

To initialize the color definition for color 11 to violet on a terminal that supports at least 12 colors, use:

init_color(11,500,0,500);

Related Information
The attroff subroutine.

584 Technical Reference, Volume 2: Base Operating System and Extensions
Curses Overview for Programming and Manipulating Video Attributes in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

cbreak, nocbreak, noraw, or raw Subroutine

Purpose
Puts the terminal into or out of CBREAK mode.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
int cbreak(void);
int nocbreak(void);
int noraw(void);
int raw(void);

Description
The cbreak subroutine sets the input mode for the current terminal to cbreak mode and overrides a call to the raw subroutine.

The nocbreak subroutine sets the input mode for the current terminal to Cooked Mode without changing the state of the ISIG and IXON flags.

The noraw subroutine sets the input mode for the current terminal to Cooked Mode and sets the ISIG and IXON flags.

The raw subroutine sets the input mode for the current terminal to Raw Mode.

Return Values
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

Examples
For the cbreak and nocbreak subroutines:
1. To put the terminal into CBREAK mode, enter:
   cbreak();
2. To take the terminal out of CBREAK mode, enter:
   nocbreak();
3. To place the terminal into raw mode, use:
   raw();
4. To place the terminal out of raw mode, use:
   noraw();

For the noraw and raw subroutines:
1. To place the terminal into raw mode, use:
   raw();
2. To place the terminal out of raw mode, use:
   noraw();
Related Information
The `getch` subroutine.

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Understanding Terminals with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

clear, erase, wclear or werase Subroutine

Purpose
Clears a window.

Library
Curses Library (`libcurses.a`)

Syntax
```c
#include <curses.h>
int clear(void);
int erase(void);

int wclear(WINDOW *win);
int werase(WINDOW *win);
```

Description
The `clear`, `erase`, `wclear`, and `werase` subroutines clear every position in the current or specified window.

The `clear` and `wclear` subroutines also achieve the same effect as calling the `clearok` subroutine, so that the window is cleared completely on the next call to the `wrefresh` subroutine for the window and is redrawn in its entirety.

Parameters

* `*win` Specifies the window to clear.

Return Values
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

Examples
For the `clear` and `wclear` subroutines:
1. To clear `stdscr` and set a clear flag for the next call to the `refresh` subroutine, enter:
   ```c
   clear();
   ```
2. To clear the user-defined window `my_window` and set a clear flag for the next call to the `wrefresh` subroutine, enter:
WINDOW *my_window;
wclear(my_window);
waddstr (my_window, "This will be cleared.");
wrefresh (my_window);

3. To erase the standard screen structure, enter:
erase();

4. To erase the user-defined window my_window, enter:
WINDOW *my_window;
werase (my_window);

Note: After the wrefresh, the window will be cleared completely. You will not see the string "This will be cleared."

For the erase and werase subroutines:
1. To erase the standard screen structure, enter:
erase();
2. To erase the user-defined window my_window, enter:
WINDOW *my_window;
werase(my_window);

Related Information
The doupdate ("doupdate, refresh, wnoutrefresh, or wrefresh Subroutines" on page 707) subroutine, erase ("erase or werase Subroutine" on page 605) and werase ("erase or werase Subroutine" on page 605) subroutines, clearok ("clearok, idlok, leaveok, scrollok, setscrreg or wsetscrreg Subroutine") subroutine, refresh ("refresh or wrefresh Subroutine" on page 658) subroutine.


clearok, idlok, leaveok, scrollok, setscrreg or wsetscrreg Subroutine

Purpose
Terminal output control subroutines.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

int clearok(WINDOW *win,
bool bf);

int idlok(WINDOW *win,
bool bf);

int leaveok(WINDOW *win,
bool bf);

int scrollok(WINDOW *win,
bool bf);

int setscrreg(int top,
int bot);
These subroutines set options that deal with output within Curses.

The **clearok** subroutine assigns the value of \( bf \) to an internal flag in the specified window that governs clearing of the screen during a refresh. If, during a refresh operation on the specified window, the flag in **curscr** is TRUE or the flag in the specified window is TRUE, then the implementation clears the screen, redraws it in its entirety, and sets the flag to FALSE in **curscr** and in the specified window. The initial state is unspecified.

The **idlok** subroutine specifies whether the implementation may use the hardware insert-line, delete-line, and scroll features of terminals so equipped. If \( bf \) is TRUE, use of these features is enabled. If \( bf \) is FALSE, use of these features is disabled and lines are instead redrawn as required. The initial state is FALSE.

The **leaveok** subroutine controls the cursor position after a refresh operation. If \( bf \) is TRUE, refresh operations on the specified window may leave the terminal's cursor at an arbitrary position. If \( bf \) is FALSE, then at the end of any refresh operation, the terminal's cursor is positioned at the cursor position contained in the specified window. The initial state is FALSE.

The **scrollok** subroutine controls the use of scrolling. If \( bf \) is TRUE, then scrolling is enabled for the specified window, with the consequences discussed in Truncation, Wrapping and Scrolling on page 28. If \( bf \) is FALSE, scrolling is disabled for the specified window. The initial state is FALSE.

The **setscreg** and **wsetscreg** subroutines define a software scrolling region in the current or specified window. The **top** and **bot** arguments are the line numbers of the first and last line defining the scrolling region. (Line 0 is the top line of the window.) If this option and the **scrollok** subroutine are enabled, an attempt to move off the last line of the margin causes all lines in the scrolling region to scroll one line in the direction of the first line. Only characters in the window are scrolled. If a software scrolling region is set and the **scrollok** subroutine is not enabled, an attempt to move off the last line of the margin does not reposition any lines in the scrolling region.

### Parameters

The parameters for the **clearok** subroutine are:

- **Flag** Sets the window clear flag. If TRUE, curses clears the window on the next call to the **wrefresh** or **refresh** subroutines. If FALSE, curses does not clear the window.
- **Window** Specifies the window to clear.

The parameters for the **idlok** subroutine are:

- **Flag** Specifies whether to enable curses to use the hardware insert/delete line feature (TRUE) or not (FALSE).
- **Window** Specifies the window it will affect.

The parameters for the **leaveok** subroutine are:

- **Flag** Specifies whether to leave the physical cursor alone after a refresh (TRUE) or to move the physical cursor to the logical cursor after a refresh (FALSE).
- **Window** Specifies the window for which to set the **Flag** parameter.
The parameters for the `scrollok` subroutine are:

Flag  Enables scrolling when set to TRUE. Otherwise, set the Flag parameter to FALSE to disable scrolling.
Window Identifies the window in which to enable or disable scrolling.

The parameters for the `setscrreg` and `wsetscrreg` subroutines are:

Bmargin Specifies the last line number in the scrolling region.
Tmargin Specifies the first line number in the scrolling region (0 is the top line of the window.).
Window Specifies the window in which to place the scrolling region. You specify this parameter only with the `wsetscrreg` subroutine.

Return Values
Upon successful completion, the `setscrreg` and `wsetscrreg` subroutines return OK. Otherwise, they return ERR.

The other subroutines always return OK.

Examples
Examples for the `clearok` subroutine are:
1. To set the user-defined screen `my_screen` to clear on the next call to the `wrefresh` subroutine, enter:
   ```
   WINDOW *my_screen;
   clearok(my_screen, TRUE);
   ```
2. To set the standard screen structure to clear on the next call to the `refresh` subroutine, enter:
   ```
   clearok(stdscr, TRUE);
   ```

Examples for the `idlok` subroutine are:
1. To enable curses to use the hardware insert/delete line feature in `stdscr`, enter:
   ```
   idlok(stdscr, TRUE);
   ```
2. To force curses not to use the hardware insert/delete line feature in the user-defined window `my_window`, enter:
   ```
   idlok(my_window, FALSE);
   ```

Examples for the `leaveok` subroutine are:
1. To move the physical cursor to the same location as the logical cursor after refreshing the user-defined window `my_window`, enter:
   ```
   WINDOW *my_window;
   leaveok(my_window, FALSE);
   ```
2. To leave the physical cursor alone after refreshing the user-defined window `my_window`, enter:
   ```
   WINDOW *my_window;
   leaveok(my_window, TRUE);
   ```

Examples for the `scrollok` subroutine are:
1. To turn scrolling on in the user-defined window `my_window`, enter:
   ```
   WINDOW *my_window;
   scrollok(my_window, TRUE);
   ```
2. To turn scrolling off in the user-defined window `my_window`, enter:
   ```
   WINDOW *my_window;
   scrollok(my_window, FALSE);
   ```

Examples for the `setscrreg` or `wsetscrreg` subroutine are:
1. To set a scrolling region starting at the 10th line and ending at the 30th line in the stdscr, enter:
   ```c
   setscreg(9, 29);
   ```
   **Note:** Zero is always the first line.

2. To set a scrolling region starting at the 10th line and ending at the 30th line in the user-defined window `my_window`, enter:
   ```c
   WINDOW *my_window;
   wsetscreg(my_window, 9, 29);
   ```

### Related Information
The `doupdate` ("doupdate, refresh, wnoutrefresh, or wrefresh Subroutines" on page 707) subroutine, `scrl` ("scrl, scroll, wscrl Subroutine" on page 668) subroutine, `refresh` or `wrefresh` ("refresh or wrefresh Subroutine" on page 658) subroutine.

**clrtobot or wclrtobot Subroutine**

**Purpose**
Erases the current line from the logical cursor position to the end of the window.

**Library**
Curses Library (libcurses.a)

**Syntax**
```c
#include <curses.h>
int clrtobot(void);

int wclrtobot(WINDOW *win);
```

**Description**
The `clrtobot` and `wclrtobot` subroutines erase all lines following the cursor in the current or specified window, and erase the current line from the cursor to the end of the line, inclusive. These subroutines do not update the cursor.

**Parameters**
*win  Specifies the window in which to erase lines.

**Return Values**
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

**Examples**
1. To erase the lines below and to the right of the logical cursor in the stdscr, enter:
   ```c
   clrtobot();
   ```

2. To erase the lines below and to the right of the logical cursor in the user-defined window `my_window`, enter:
   ```c
   WINDOW *my_window;
   wclrtobot(my_window);
   ```
 clrtoeol or wclrtoeol Subroutine

Purpose
Erases the current line from the logical cursor position to the end of the line.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
int clrtoeol(void);

int wclrtoeol(WINDOW *win);

Description
The clrtoeol and wclrtoeol subroutines erase the current line from the cursor to the end of the line, inclusive, in the current or specified window. These subroutines do not update the cursor.

Parameters
*win Specifies the window in which to clear the line.

Return Values
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

Examples
1. To clear the line to the right of the logical cursor in the stdscr, enter:
   clrtoeol();
2. To clear the line to the right of the logical cursor in the user-defined window my_window, enter:
   WINDOW *my_window;
   wclrtoeol(my_window);

Related Information
The doupdate subroutine.

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
color_content Subroutine

Purpose
Returns the current intensity of the red, green, and blue (RGB) components of a color.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
color_content(Color, R, G, B)
short Color;

Description
The color_content subroutine, given a color number, returns the current intensity of its red, green, and blue (RGB) components. This subroutine stores the information in the address specified by the R, G, and B arguments. If successful, this returns OK. Otherwise, this subroutine returns ERR if the color does not exist, is outside the valid range, or the terminal cannot change its color definitions.

To determine if you can change the color definitions for a terminal, use the can_change_color subroutine. You must call the start_color subroutine before you can call the color_content subroutine.

Note: The values stored at the addresses pointed to by R, G, and B are between 0 (no component) and 1000 (maximum amount of component) inclusive.

Return Values
OK Indicates the subroutine was successful.
ERR Indicates the color does not exist, is outside the valid range, or the terminal cannot change its color definitions.

Parameters
B Points to the address that stores the intensity value of the blue component.
Color Specifies the color number. The color parameter must be a value between 0 and COLORS-1 inclusive.
R Points to the address that stores the intensity value of the red component.
G Points to the address that stores the intensity value of the green component.

Example
To obtain the RGB component information for color 10 (assuming the terminal supports at least 11 colors), use:
short *r, *g, *b; color_content(10, r, g, b);
Related Information

The **start_color** subroutine.

Curses Overview for Programming

Manipulating Video Attributes

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

**copywin Subroutine**

**Purpose**

Copies a region of a window.

**Library**

Curses Library (**libcurses.a**)  

**Syntax**

```c
#include <curses.h>

int copywin(const WINDOW *scrwin,
            WINDOW *dstwin,
            int sminrow,
            int smincol,
            int dminrow,
            int dmincol,
            int dmaxrow,
            int dmaxcol,
            int overlay);
```

**Description**

The **copywin** subroutine provides a finer granularity of control over the **overlay** and **overwrite** subroutines. As in the **prefresh** subroutine, a rectangle is specified in the destination window, \((dminrow, dmincol)\) and \((dmaxrow, dmaxcol)\), and the upper-left-corner coordinates of the source window, \((sminrow, smincol)\). If the **overlay** subroutine is TRUE, then copying is non-destructive, as in the **overlay** subroutine. If the **overlay** subroutine is FALSE, then copying is destructive, as in the **overwrite** subroutine.

**Parameters**

- **srcwin** Points to the source window containing the region to copy.
- **dstwin** Points to the destination window to copy into.
- **sminrow** Specifies the upper left row coordinate of the source region.
- **smincol** Specifies the upper left column coordinate of the source region.
- **dminrow** Specifies the upper left row coordinate of the destination region.
- **dmincol** Specifies the upper left column coordinate for the destination region.
- **dmaxrow** Specifies the lower right row coordinate for the destination region.
- **dmaxcol** Specifies the lower right column coordinate for the destination region.
- **overlay** Sets the type of copy. If set to TRUE the copy is non-destructive. Otherwise, if set to FALSE, the copy is destructive.

**Return Values**

Upon successful completion, the **copywin** subroutine returns **OK**. Otherwise, it returns **ERR**.
Examples
To copy to an area in the destination window defined by coordinates (30,40), (30,49), (39,40), and (39,49) beginning with coordinates (0,0) in the source window, enter the following:

```
WINDOW *srcwin, *dstwin;

copywin(srcwin, dstwin,
    0, 0, 30, 40, 39, 49,
    TRUE);
```

The example copies ten rows and ten columns from the source window beginning with coordinates (0,0) to the region in the destination window defined by the upper left coordinates (30, 40) and lower right coordinates (39, 49). Because the Overlay parameter is set to TRUE, the copy is nondestructive and blanks from the source window are not copied.

Related Information
The newpad ([“newpad, pnonrefresh, prefresh, or subpad Subroutine” on page 642](#)) and overlay or overwrite ([“overlay or overwrite Subroutine” on page 651](#)) subroutines.

---

curs_set Subroutine

### Purpose
Sets the cursor visibility.

### Library
Curses Library ([libcurses.a](#))

### Syntax

```
#include <curses.h>

int curs_set(int visibility);
```

### Description
The **curs_set** subroutine sets the appearance of the cursor based on the value of visibility:

<table>
<thead>
<tr>
<th>Value of visibility</th>
<th>Appearance of Cursor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>invisible</td>
</tr>
<tr>
<td>1</td>
<td>terminal-specific normal mode</td>
</tr>
<tr>
<td>2</td>
<td>terminal-specific high visibility mode</td>
</tr>
</tbody>
</table>

The terminal does not necessarily support all the above values.
Parameters

Visibility

Sets the cursor state. You can set the cursor state to one of the following:

0  Invisible
1  Visible
2  Very visible

Return Values

If the terminal supports the cursor mode specified by visibility, then the cur_set subroutine returns the previous cursor state. Otherwise, the subroutine returns ERR.

Examples

To set the cursor state to invisible, use:

curs_set(0);

Related Information

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs

Setting Video Attributes in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs

---

def_prog_mode, def_shell_mode, reset_prog_mode or reset_shell_mode Subroutine

Purpose

Saves/restores the program or shell terminal modes.

Library

Curses Library (libcurses.a)

Syntax

#include <curses.h>

int def_prog_mode (void);

int def_shell_mode (void);

int reset_prog_mode (void);

int reset_shell_mode (void);
Description
The `def_prog_mode` subroutine saves the current terminal modes as the “program” (in Curses) state for use by the `reset_prog_mode` subroutine.

The `def_shell_mode` subroutine saves the current terminal modes as the “shell” (not in Curses) state for use by the `reset_shell_mode` subroutine.

The `reset_prog_mode` subroutine restores the terminal to the “program” (in Curses) state.

The `reset_shell_mode` subroutine restores the terminal to the “shell” (not in Curses) state.

These subroutines affect the mode of the terminal associated with the current screen.

Return Values
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

Examples
For the `def_prog_mode` subroutine:

To save the "in curses" state, enter:

```
def_prog_mode();
```

For the `def_shell_mode` subroutine:

To save the "out of curses" state, enter:

```
def_shell_mode();
```

This routine saves the "out of curses" state.

Related Information
The `doupdate` ("doupdate, refresh, wnoutrefresh, or wrefresh Subroutines" on page 707), `endwin` ("endwin Subroutine" on page 604), `initscr` ("initscr and newterm Subroutine" on page 627), and the `setupterm` ("setupterm Subroutine" on page 674) subroutines.

def_shell_mode Subroutine

Purpose
Saves the current terminal modes as shell mode ("out of curses").

Library
Curses Library (libcurses.a)

Syntax
```
#include <curses.h>
def_shell_mode();
```
Description
The `def_shell_mode` subroutine saves the current terminal driver line discipline modes in the current terminal structure for later use by `reset_shell_mode()`. The `def_shell_mode` subroutine is called automatically by the `setupterm` subroutine.

This routine would normally not be called except by a library routine.

Example
To save the “out of curses” state, enter:
```c
def_shell_mode();
```
This routine saves the “out of curses” state.

Related Information
The `setupterm` subroutine.

Curses Overview for Programming List of Curses Subroutines Understanding Terminals with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

del_curterm, restartterm, set_curterm, or setupterm Subroutine

Purpose
Interfaces to the terminfo database.

Library
Curses Library (`libcurses.a`)

Syntax
```c
#include <term.h>

int del_curterm(TERMINAL *oterm);

int restartterm(char *term,
int fildes,
int *erret);

TERMINAL *set_curterm(TERMINAL *nterm);

int setupterm(char *term,
int fildes,
int *erret);
```

Description
The `del_curterm`, `restartterm`, `set_curterm`, `setupterm` subroutines retrieve information from the terminfo database.

To gain access to the terminfo database, the `setupterm` subroutine must be called first. It is automatically called by the `initscr` and `newterm` subroutines. The `setupterm` subroutine initialises the other subroutines to use the terminfo record for a specified terminal (which depends on whether the `use_env` subroutine was called). It sets the `dur_term` external variable to a TERMINAL structure that contains the record from the terminfo database for the specified terminal.

The terminal type is the character string `term`; if `term` is a null pointer, the environment variable `TERM` is used. If `TERM` is not set or if its value is an empty string, the “unknown” is used as the terminal type. The
application must set the *fildes parameter to a file descriptor, open for output, to the terminal device, before calling the *setupterm subroutine. If the *errret parameter is not null, the integer it points to is set to one of the following values to report the function outcome:

-1 The terminfo database was not found (function fails).
0 The entry for the terminal was not found in *terminfo (function fails).
1 Success.

A simple call to the *setupterm subroutine that uses all the defaults and sends the output to stdout is:

```c
setupterm(char *), fileno(stdout), (int *)0);
```

The *set_curterm subroutine sets the variable *cur_term to *nterm, and makes all of the terminfo boolean, numeric, and string variables use the values from *nterm.

The *del_curterm subroutine frees the space pointed to by *oterm and makes it available for further use. If *oterm is the same as *cur_term, references to any of the terminfo boolean, numeric, and string variables thereafter may refer to invalid memory locations until the *setupterm subroutine is called again.

The *restartterm subroutine assumes a previous call to the *setupterm subroutine (perhaps from the *initscr or *newterm subroutine). It lets the application specify a different terminal type in *term and updates the information returned by the *baudrate subroutine based on the *fildes parameter, but does not destroy other information created by the *initscr, *newterm, or *setupterm subroutines.

**Parameters**

*oterm
*term
*fildes
*errret
*nterm

**Return Values**

Upon successful completion, the *set_curterm subroutine returns the previous value of *cur_term. Otherwise, it returns a null pointer.

Upon successful completion, the other subroutines return OK. Otherwise, they return ERR.

**Examples**

To free the space occupied by a TERMINAL structure called *my_term, use:

```c
TERMINAL *my_term; del_curterm(my_term);
```

For the *restartterm subroutine:

To restart an aixterm after a previous memory save and exit on error with a message, enter:

```c
restartterm("aixterm", 1, (int*)0);
```

For the *set_curterm subroutine:

To set the *cur_term variable to point to the *my_term terminal, use:
For the `setupterm` subroutine:

To determine the current terminal's capabilities using `$TERM` as the terminal name, standard output as output, and returning no error codes, enter:

```c
setupterm((char*) 0, 1, (int*) 0);
```

**Related Information**

The `baudrate` subroutine, `longname` subroutine, `putc` subroutine, `tgetent` subroutine, `tigetflag` subroutine.

**delay_output Subroutine**

**Purpose**
Set the delay output.

**Library**
Curses Library (`libcurses.a`)

**Syntax**
```
#include <curses.h>

int delay_output(int ms);
```

**Description**
On terminals that support pad characters, the `delay_output` subroutine pauses the output for at least `ms` milliseconds. Otherwise, the length of the delay is unspecified.

**Parameters**

- `ms` Specifies the number of milliseconds to delay output.

**Return Values**
Upon successful completion, the `delay_output` subroutine returns OK. Otherwise, it returns ERR.
Examples
To set the output to delay 250 milliseconds, enter:
delay_output(250);

Related Information
[Curses Overview for Programming] [List of Curses Subroutines] [Understanding Terminals with Curses] in
AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

delch, mvdelch, mvwdelch or wdelch Subroutine

Purpose
Deletes the character from a window.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
int delch(void);
int mvdelch
(int y
int x);

mvwdelch
(WINDOW *win)
int y
int x);

wdelch
(WINDOW *win);

Description
The delch, mvdelch, mvwdelch, and wdelch subroutines delete the character at the current or specified position in the current or specified window. This subroutine does not change the cursor position.

Parameters
x
y
*win Identifies the window from which to delete the character.

Return Values
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

Examples
1. To delete the character at the current cursor location in the standard screen structure, enter:
   mvdelch();
2. To delete the character at cursor position y=20 and x=30 in the standard screen structure, enter:
   mvwdelch(20, 30);
3. To delete the character at cursor position y=20 and x=30 in the user-defined window my_window, enter:
Related Information
Curses Overview for Programming List of Curses Subroutines Manipulating Characters with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

deleteln or wdeleteln Subroutine

Purpose
Deletes lines in a window.

Library
Curses Library (libcurses.a)

Syntax
```c
#include <curses.h>
int deleteln(void);

int wdeleteln(WINDOW *win);
```

Description
The deleteln and wdeleteln subroutines delete the line containing the cursor in the current or specified window and move all lines following the current line one line toward the cursor. The last line of the window is cleared. The cursor position does not change.

Parameters
*win Specifies the window in which to delete the line.

Return Values
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

Examples
1. To delete the current line in stdscr, enter:
   `deleteln();`
2. To delete the current line in the user-defined window my_window, enter:
   ```c
   WINDOW *my_window;
   wdeleteln(my_window);
   ```

Related Information
Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Manipulating Characters with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
delwin Subroutine

Purpose
Deletes a window.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

int delwin(WINDOW *win);

Description
The delwin subroutine deletes win, freeing all memory associated with it. The application must delete subwindows before deleting the main window.

Parameters
*win Specifies the window to delete.

Return Values
Upon successful completion, the delwin subroutine returns OK. Otherwise, it returns ERR.

Examples
To delete the user-defined window my_window and its subwindow my_sub_window, enter:

WINDOW *my_sub_window,*my_window;
delwin(my_sub_window);
delwin(my_window);

Related Information
The derwin (derwin, newwin, or subwin Subroutine” on page 646) subroutine.

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Manipulating Window Data with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

echo or noecho Subroutine

Purpose
Enables/disables terminal echo.

Library
Curses Library (libcurses.a)
Syntax

```c
#include <curses.h>
int echo(void);
int noecho(void);
```

Description

The `echo` subroutine enables Echo mode for the current screen. The `noecho` subroutine disables Echo mode for the current screen. Initially, curses software echo mode is enabled and hardware echo mode of the tty driver is disabled. The `echo` and `noecho` subroutines control software echo only. Hardware echo must remain disabled for the duration of the application, else the behaviour is undefined.

Return Values

Upon successful completion, these subroutines return `OK`. Otherwise, they return `ERR`.

Examples

1. To turn echoing on, use:
   ```c
echo();
```
2. To turn echoing off, use:
   ```c
noecho();
```

Related Information

The `wgetch` subroutine is equivalent to a call to the `addch` subroutine followed by a call to the `refresh` subroutine. The `wechochar` subroutine is equivalent to a call to the `waddch` subroutine followed by a call to the `wrefresh` subroutine.

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines and Understanding Terminals with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

**echochar or wechochar Subroutines**

**Purpose**

Echos single-byte character and rendition to a window and refreshes the window.

**Library**

Curses Library (`libcurses.a`)

**Syntax**

```c
#include <curses.h>
int echochar(const chtype ch);
int wechochar(WINDOW *win, const chtype ch);
```

**Description**

The `echochar` subroutine is equivalent to a call to the `addch` subroutine followed by a call to the `refresh` subroutine. The `wechochar` subroutine is equivalent to a call to the `waddch` subroutine followed by a call to the `wrefresh` subroutine.
**Return Values**
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

**Example**
To output the character I to the stdscr at the present cursor location and to update the physical screen, do the following:
```c
echochar('I');
```

**Related Information**
The `addch`, `doupdate`, `echo_wchar`, `waddch`, `wmvaddch`, and `mvaddch` (or `addch`, `mvaddch`, `mwaddch`, `waddch` Subroutine” on page 573) subroutines.

Curses Overview for Programming]] and List of Curses Subroutines]] in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Manipulating Characters with Curses]] in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

**endwin Subroutine**

**Purpose**
Suspends curses session.

**Library**
Curses Library (libcurses.a)

**Syntax**
```c
#include <curses.h>

int endwin(void)
```

**Description**
The `endwin` subroutine restores the terminal after Curses activity by at least restoring the saved shell terminal mode, flushing any output to the terminal and moving the cursor to the first column of the last line of the screen. Refreshing a window resumes program mode. The application must call the `endwin` subroutine for each terminal being used before exiting. If the `newterm` subroutine is called more than once for the same terminal, the first screen created must be the last one for which the `endwin` subroutine is called.

**Return Values**
Upon successful completion, the `endwin` subroutine returns OK. Otherwise, it returns ERR.

**Examples**
To terminate curses permanently or temporarily, enter:
```c
endwin();
```

**Related Information**
The `doupdate` (“doupdate, refresh, wnoutrefresh, or wrefresh Subroutines” on page 707) subroutine, `initscr` (“initscr and newterm Subroutine” on page 627) subroutine, and `isendwin` subroutine.
erase or werase Subroutine

Purpose
Copies blank spaces to every position in a window.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
erase()
werase(

WINDOW *Window;

Description
The erase and werase subroutines copy blank spaces to every position in the specified window. Use the erase subroutine with the stdscr and the werase subroutine with user-defined windows.

Parameters
Window Specifies the window to erase.

Examples
1. To erase the standard screen structure, enter:
   erase();
2. To erase the user-defined window my_window, enter:
   WINDOW *my_window;
   werase(my_window);

Related Information
Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Manipulating Characters with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
erasechar, erasewchar, killchar, and killwchar Subroutine

Purpose
Terminal environment query functions.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

char erasechar(void);

int erasewchar(wchar_t *ch);

char killchar(void);

int killwchar(wchar_t *ch);

Description
The erasechar subroutine returns the current character, chosen by the user. The erasechar subroutine stores the current erase character in the object pointed to by the ch parameter. If no erase character has been defined, the subroutine will fail and the object pointed to by ch will not be changed.

The killchar subroutine returns the current line.

The killchar subroutine stores the current line kill character in the object pointed to by ch. If no line kill character has been defined, the subroutine will fail and the object pointed to by ch will not be changed.

Return Values
The erasechar subroutine returns the erase character and the killchar subroutine returns the line kill character. The return value is unspecified when these characters are multi-byte characters.

Upon successful completion, the erasechar subroutine and the killchar subroutine return OK. Otherwise, they return ERR.

Examples
To retrieve a user's erase character and return it to the user-defined variable myerase, enter:

myerase = erasechar();

Related Information
The clearok subroutine, tcgetattr subroutine.

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
filter Subroutine

Purpose
Disables use of certain terminal capabilities.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
void filter(void);

Description
The filter subroutine changes the algorithm for initialising terminal capabilities that assume that the terminal has more than one line. A subsequent call to the initscr or newterm subroutine performs the following actions:

- Disables use of clear, cud, cud1, cup, cuu1, and vpa.
- Sets the value of the home string to the value of the cr. string.
- Sets lines equal to 1.

Any call to the filter subroutine must precede the call to the initscr or newterm subroutine.

Related Information
The initscr (“initscr and newterm Subroutine” on page 627) subroutine, newterm (“newterm Subroutine” on page 644) subroutine.

Curses Overview for Programming and List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

flash Subroutine

Purpose
Flashes the screen.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
int flash(void);

Description
The flash subroutine alerts the user. It flashes the screen, or if that is not possible, it sounds the audible alarm on the terminal. If neither signal is possible, nothing happens.

Return Values
The flash subroutine always returns OK.
Examples
To cause the terminal to flash, enter:
flash();

Related Information
The beep subroutine.

flushinp Subroutine

Purpose
Discards input.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
int flushinp(void);

Description
The flushinp subroutine discards (flushes) any characters in the input buffers associated with the current screen.

Return Values
The flushinp subroutine always returns OK.

Examples
To flush all type-ahead characters typed by the user but not yet read by the program, enter:
flushinp();

Related Information
In AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.:

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
garbagedlines Subroutine

Purpose
Discards and replaces a number of lines in a window.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

garbagedlines(Window, BegLine, NumLines)
WINDOW * Window;
int BegLine, NumLines;

Description
The garbagedlines subroutine discards and replaces lines in a window. The Begline parameter specifies the beginning line number and the Numlines parameter specifies the number of lines to discard. Curses discards and replaces the specified lines before adding more data.

Uses this subroutine for applications that need to redraw a line that is garbled. Lines may become garbled as the result of noisy communication lines. Instead of refreshing the entire display, use the garbagedlines subroutine to refresh a portion of the display and to avoid even more communication noise.

Parameters
Window        Points to a window.
BegLine       Identifies the beginning line in a range of lines to discard.
NumLines      Specifies the total number of lines in a range of lines to discard and replace.

Examples
To discard and replace 5 lines in the mywin window starting with line 10, use:

WINDOW *mywin; garbagedlines(mywin, 10, 5);

Related Information
Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
Manipulating Window Data with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
getbegyx, getmaxyx, getparyx, or getyx Subroutine

 Purpose
 Gets the cursor and window coordinates.

 Library
 Curses Library (libcurses.a)

 Syntax

```
#include <curses.h>

void getbegyx(WINDOW *win,
        int y,
        int x);

void getmaxyx(WINDOW *win,
        int y,
        int x);

void getparyx(WINDOW *win,
        int y,
        int x);

void getyx(WINDOW *win,
        int y,
        int x);
```

 Description
 The `getbegyx` macro stores the absolute screen coordinates of the specified window's origin in `y` and `x`.

 The `getmaxyx` macro stores the number of rows of the specified window in `y` and `x` and stores the window's number of columns in `x`.

 The `getparyx` macro, if the specified window is a subwindow, stores in `y` and `x` the coordinates of the window's origin relative to its parent window. Otherwise, -1 is stored in `y` and `x`.

 The `getyx` macro stores the cursor position of the specified window in `y` and `x`.

 Parameters

- `*win` Identifies the window to get the coordinates from.
- `Y` Returns the row coordinate.
- `X` Returns the column coordinate.

 Examples
 For the `getbegyx` subroutine:

 To obtain the beginning coordinates for the `my_win` window and store in integers `y` and `x`, use:

```
WINDOW *my_win;
int y, x;
getbegyx(my_win, y, x);
```

 For the `getmaxyx` subroutine:

 To obtain the size of the `my_win` window, use:

```
```
WINDOW *my_win;

int y,x;
getmaxyx(my_win, y, x);

Integers y and x will contain the size of the window.

Related Information

Controlling the Cursor with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

getch, mvgetch, mvwgetch, or wgetch Subroutine

Purpose

Gets a single-byte character from the terminal.

Library

Curses Library (libcurses.a)

Syntax

#include <curses.h>

int getch(void)

int mvgetch(int y,
int x);

int mvwgetch(WINDOW *win,
int y,
int x);

int wgetch(WINDOW *win);

Description

The getch, wgetch, mvgetch, and mvwgetch subroutines read a single-byte character from the terminal associated with the current or specified window. The results are unspecified if the input is not a single-byte character. If the keypad subroutine is enabled, these subroutines respond to the corresponding KEY_ value defined in <curses.h>.

Processing of terminal input is subject to the general rules described in Section 3.5 on page 34.

If echoing is enabled, then the character is echoed as though it were provided as an input argument to the addch subroutine, except for the following characters:
<backspace>,
<left-arrow> and
the current erase character:
The input is interpreted as specified in Section 3.4.3 on page 31 and then the character at the resulting cursor position is deleted as though the delch subroutine was called, except that if the cursor was originally in the first column of the line, then the user is alerted as though the beep subroutine was called.

The user is alerted as though the beep subroutine was called. Information concerning the function keys is not returned to the caller.

Function Keys
If the current or specified window is not a pad, and it has been moved or modified since the last refresh operation, then it will be refreshed before another character is read.

The Importance of Terminal Modes
The output of the getch subroutines is, in part, determined by the mode of the terminal. The following describes the action of the getch subroutines in each type of terminal mode:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Action of getch Subroutines</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODELAY</td>
<td>Returns a value of ERR if there is no input waiting.</td>
</tr>
<tr>
<td>DELAY</td>
<td>Halts execution until the system passes text through the program. If CBREAK mode is also set, the program stops after receiving one character. If NOCBREAK mode is set, the getch subroutine stops reading after the first new line character.</td>
</tr>
<tr>
<td>HALF-DELAY</td>
<td>Halts execution until a character is typed or a specified time out is reached. If echo is set, the character is also echoed to the window.</td>
</tr>
</tbody>
</table>

Note: When using the getch subroutines do not set both the NOCBREAK mode and the ECHO mode at the same time. This can cause undesirable results depending on the state of the tty driver when each character is typed.

Getting Function Keys
If your program enables the keyboard with the keypad subroutine, and the user presses a function key, the token for that function key is returned instead of raw characters. The possible function keys are defined in the /usr/include/curses.h file. Each #define macro begins with a KEY_ prefix.

If a character is received that could be the beginning of a function key (such as an Escape character) curses sets a timer. If the remainder of the sequence is not received before the timer expires, the character is passed through. Otherwise, the function key’s value is returned. For this reason, after a user presses the Esc key there is a delay before the escape is returned to the program. Programmers should not use the Esc key for a single character routine.

Within the getch subroutine, a structure of type timeval, defined in the /usr/include/sys/time.h file, indicates the maximum number of microseconds to wait for the key response to complete.

The ESCDELAY environment variable sets the length of time to wait before timing out and treating the ESC keystroke as the ESC character rather than combining it with other characters in the buffer to create a key sequence. The ESCDELAY environment variable is measured in fifths of a millisecond. If ESCDELAY is 0, the system immediately composes the ESCAPE response without waiting for more information from the buffer. The user may choose any value between 0 and 99,999, inclusive. The default setting for the ESCDELAY environment variable is 500 (one tenth of a second).

Programs that do not want the getch subroutines to set a timer can call the notimeout subroutine. If notimeout is set to TRUE, curses does not distinguish between function keys and characters when retrieving data.

The getch subroutines might not be able to return all function keys because they are not defined in the terminfo database or because the terminal does not transmit a unique code when the key is pressed. The following function keys may be returned by the getch subroutines:
KEY_MIN          Minimum curses key.
KEY_BREAK        Break key (unreliable).
KEY_DOWN         Down Arrow key.
KEY_UP           Up Arrow key.
KEY_LEFT         Left Arrow key.
KEY_RIGHT        Right Arrow key.
KEY_HOME         Home key.
KEY_BACKSPACE    Backspace.
KEY_F(n)         Function key Fn, where n is an integer from 0 to 64.
KEY_DL           Delete line.
KEY_IL           Insert line.
KEY_DC           Delete character.
KEY_IC           Insert character or enter insert mode.
KEY_EIC          Exit insert character mode.
KEY_CLEAR        Clear screen.
KEY_EOS          Clear to end of screen.
KEY_EOL          Clear to end of line.
KEY_SF           Scroll 1 line forward.
KEY_SR           Scroll 1 line backwards (reverse).
KEY_NPAGE        Next page.
KEY_PPAGE        Previous page.
KEY_STAB         Set tab.
KEY_CATAB        Clear all tabs.
KEY_ENTER        Enter or send (unreliable).
KEY_SRESET       Soft (partial) reset (unreliable).
KEY_RESET        Reset or hard reset (unreliable).
KEY_PRINT        Print or copy.
KEY_LL           Home down or bottom (lower left).
KEY_A1           Upper-left key of keypad.
KEY_A3           Upper-right key of keypad.
KEY_B2           Center-key of keypad.
KEY_C1           Lower-left key of keypad.
KEY_C3           Lower-right key of keypad.
KEY_BTAB         Back tab key.
KEY_BEG          beg(inning) key
KEY_CANCEL       cancel key
KEY_CLOSE        close key
KEY_COMMAND      cmd (command) key
KEY_COPY         copy key
KEY_CREATE       create key
KEY_END          end key
KEY_EXIT         exit key
KEY_FIND         find key
KEY_HELP         help key
KEY_MARK         mark key
KEY_MESSAGE      message key
KEY_MOVE         move key
KEY_NEXT         next object key
KEY_OPEN         open key
KEY_OPTIONS      options key
KEY_PREVIOUS     previous object key
KEY_REDO         redo key
KEY_REFERENCE    ref(ERENCE) key
KEY_REFRESH refresh key
KEY_REPLACE replace key
KEY_RESTART restart key
KEY_RESUME resume key
KEY_SAVE save key
KEY_SBEG shifted beginning key
KEY_SCANCEL shifted cancel key
KEY_SCOMMAND shifted command key
KEY_SCOPY shifted copy key
KEY_SCREATE shifted create key
KEY_SDC shifted delete char key
KEY(SDL) shifted delete line key
KEY_SELECT select key
KEY_SEND shifted end key
KEY_SEOL shifted clear line key
KEY_SEXIT shifted exit key
KEY_SFINISH shifted find key
KEY_SHELPEEPEPEPP shifted help key
KEY_SHOME shifted home key
KEY_SIC shifted input key
KEY_SLEFT shifted left arrow key
KEY_SMESSAGE shifted message key
KEY_SMOVE shifted move key
KEY_SNEXT shifted next key
KEY_SOPTIONS shifted options key
KEY_SPREVIOUS shifted prev key
KEY_SPRINT shifted print key
KEY_SREDO shifted redo key
KEY_SREPLACE shifted replace key
KEY_SRIGHT shifted right arrow
KEY_SRSUME shifted resume key
KEY_SSAVE shifted save key
KEY_SSUSPEND shifted suspend key
KEY_SUNDO shifted undo key
KEY_SUSPEND suspend key
KEY_UNDO undo key

Parameters

Column Specifies the horizontal position to move the logical cursor to before getting the character.
Line Specifies the vertical position to move the logical cursor to before getting the character.
Window Identifies the window to get the character from and echo it into.

Return Values

Upon successful completion, the `getch`, `mvwgetch`, and `wgetch` subroutines, CURSES, and Curses Interface return the single-byte character, KEY_ value, or ERR. When in the nodelay mode and no data is available, ERR is returned.

Examples

1. To get a character and echo it to the stdscr, use:
   ```c
   mvgetch();
   ```
2. To get a character and echo it into stdscr at the coordinates y=20, x=30, use:
mvgetch(20, 30);

3. To get a character and echo it into the user-defined window `my_window` at coordinates y=20, x=30, use:
   ```c
   WINDOW *my_window;
   mvwgetch(my_window, 20, 30);
   ```

Related Information
The `cbreak` ("cbreak, nocbreak, noraw, or raw Subroutine" on page 585), `doupdate` ("doupdate, refresh, wnoutrefresh, or wrefresh Subroutines" on page 707), and `insch` ("insch, mvinsch, mvwinsch, or winsch Subroutine" on page 628) subroutines, `keypad` ("keypad Subroutine" on page 632) subroutine, `meta` ("meta Subroutine" on page 638) subroutine, `nodelay` ("nodelay Subroutine" on page 648) subroutine, `echo` or `noecho` ("echo or noecho Subroutine" on page 602) subroutine, `notimeout` ("notimeout, timeout, wtimeout Subroutine" on page 649) subroutine, `ebreak` or `nocbreak` ("cbreak, nocbreak, noraw, or raw Subroutine" on page 585) subroutine.

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Manipulating Characters with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

getmaxyx Subroutine

Purpose
Returns the size of a window.

Library
Curses Library (libcurses.a)

Syntax
```c
#include <curses.h>

getmaxyx( Window, Y, X);
WINDOW *Window;
int Y, X;
```

Description
The `getmaxyx` subroutine returns the size of a window. The size is returned as the number of rows and columns in the window. The values are stored in integers `Y` and `X`.

Parameters
- `Window` Identifies the window whose size to get.
- `Y` Contains the number of rows in the window.
- `X` Contains the number of columns in the window.

Example
To obtain the size of the `my_win` window, use:
WINDOW *my_win;

int y,x;
getmaxyx(my_win, y, x);

Integers y and x will contain the size of the window.

Related Information
- Controlling the Cursor with Curses [AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs]
- Curses Overview for Programming [AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs]
- List of Curses Subroutines [AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs]

getnstr, getstr, mvgetnstr, mvgetstr, mvwgetnstr, mvwgetstr, wgetnstr, or wgetstr Subroutine

Purpose

Gets a multi-byte character string from the terminal.

Library

Curses Library (libcurses.a)

Syntax

#include <curses.h>

int getnstr(char *str, int n);
int getstr(char *str);

int mvgetnstr(int y, int x, char *str, int n);
int mvgetstr(int y, int x, char *str);

int mvwgetnstr(WINDOW *win, int y, int x, char *str, int n);
int mvwgetstr(WINDOW *win, int y, int x, char *str, int n);

616   Technical Reference, Volume 2: Base Operating System and Extensions
int x,
char *str);

int wgetnstr(WINDOW *win,
char *str,
int n);

int wgetstr(WINDOW *win,
char *str);

**Description**

The effect of the `getstr` subroutine is as though a series of calls to the `getch` subroutine was made, until a newline, carriage return, or end-of-file is received. The resulting value is placed in the area pointed to by `str`. The string is then terminated with a null byte. The `getnstr`, `mvgetnstr`, `mvwgetnstr`, and `wgetnstr` subroutines read at most `n` bytes, thus preventing a possible overflow of the input buffer. The user's erase and kill characters are interpreted, as well as any special keys (such as function keys, home key, clear key, and so on).

The `mvgetstr` subroutines is identical to the `getstr` subroutine except that it is as though it is a call to the `move` subroutine and then a series of calls to the `getch` subroutine. The `mvwgetstr` subroutine is identical to the `getstr` subroutine except that it is as though it is a call to the `wmove` subroutine and then a series of calls to the `wgetch` subroutine.

The `mvgetnstr` subroutines is identical to the `getstr` subroutine except that it is as though it is a call to the `move` subroutine and then a series of calls to the `getch` subroutine. The `mvwgetnstr` subroutine is identical to the `getstr` subroutine except that it is as though it is a call to the `wmove` subroutine and then a series of calls to the `wgetch` subroutine.

The `getstr`, `wgetstr`, `mvgetstr`, and `mvwgetstr` subroutines will only return the entire multi-byte sequence associated with a character. If the array is large enough to contain at least one character, the subroutines fill the array with complete characters. If the array is not large enough to contain any complete characters, the function fails.

**Parameters**

- `n`  
- `x`  
- `y`  
- `*str` Identifies where to store the string.  
- `*win` Identifies the window to get the string from and echo it into.

**Return Values**

Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

**Examples**

1. To get a string, store it in the user-defined variable `my_string`, and echo it into the stdscr, enter:
   ```c
   char *my_string;
   getstr(my_string);
   ```

2. To get a string, echo it into the user-defined window `my_window`, and store it in the user-defined variable `my_string`, enter:
   ```c
   WINDOW *my_window;
   char *my_string;
   wgetstr(my_window, my_string);
   ```
3. To get a string in the stdscr at coordinates y=20, x=30, and store it in the user-defined variable my_string, enter:
   
   ```
   char *string;
   mvgetstr(20, 30, string);
   ```

4. To get a string in the user-defined window my_window at coordinates y=20, x=30, and store it in the user-defined variable my_string, enter:
   
   ```
   WINDOW *my_window;
   char *my_string;
   mvwgetstr(my_window, 20, 30, my_string);
   ```

**Related Information**

The **beep** subroutine, **getch**, **mvgetch**, **mvwgetch**, or **wgetch** subroutine, **keypad** subroutine, **nodelay** subroutine, **wgetch** subroutine, **getch**, **mvgetch**, **mvwgetch**, or **wgetch** subroutine.

**getsyx Subroutine**

**Purpose**

Retrieves the current coordinates of the virtual screen cursor.

**Library**

Curses Library (libcurses.a)

**Syntax**

```
#include <curses.h>

getsyx(Y, X)
int *Y, *X
```

**Description**

The **getsyx** subroutine retrieves the current coordinates of the virtual screen cursor and stores them in the location specified by Y and X. The current coordinates are those where the cursor was placed after the last call to the **wnoutrefresh**, **pnoutrefresh**, or **wrefresh**, subroutine. If the **leaveok** subroutine was TRUE for the last window refreshed, then the **getsyx** subroutine returns -1 for both X and Y.

If lines have been removed from the top of the screen using the **ripoflone** subroutine, Y and X include these lines. Y and X should only be used as arguments for the **setsyx** subroutine.

The **getsyx** subroutine, along with the **setsyx** subroutine, is meant to be used by a user-defined function that manipulates curses windows but wants the position of the cursor to remain the same. Such a function would do the following:

- Call the **getsyx** subroutine to obtain the current virtual cursor coordinates.
- Continue manipulating the windows.
- Call the **wnoutrefresh** subroutine on each window manipulated.
- Reset the current virtual cursor coordinates to the original values with the **setsyx** subroutine.
- Refresh the display with a call to the **doupdate** subroutine.
Parameters

\( X \) Points to the current row position of the virtual screen cursor. A value of -1 indicates the `leaveok` subroutine was TRUE for the last window refreshed.

\( Y \) Points to the current column position of the virtual screen cursor. A value of -1 indicates the `leaveok` subroutine was TRUE for the last window refreshed.

Related Information

- [Curses Overview for Programming](#) in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
- [Controlling the Cursor with Curses](#) in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
- [List of Curses Subroutines](#) in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

getyx Macro

**Purpose**

Returns the coordinates of the logical cursor in the specified window.

**Library**

Curses Library (*libcurses.a*)

**Syntax**

```c
#include <curses.h>

getyx(Window, Line, Column)

WINDOW *Window;
int Line, Column;
```

**Description**

The `getyx` macro returns the coordinates of the logical cursor in the specified window.

**Parameters**

- `Window` Identifies the window to get the cursor location from.
- `Column` Holds the column coordinate of the logical cursor.
- `Line` Holds the line or row coordinate of the logical cursor.

**Example**

To get the location of the logical cursor in the user-defined window `my_window` and then put these coordinates in the user-defined integer variables `Line` and `Column`, enter:

```c
WINDOW *my_window;
int line, column;
getyx(my_window, line, column);
```
Related Information

- **Controlling the Cursor with Curses** in *AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.*
- **Curses Overview for Programming** in *AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.*
- **List of Curses Subroutines** in *AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.*

---

### halfdelay Subroutine

#### Purpose
Controls input character delay mode.

#### Library
Curses Library (*libcurses.a*)

#### Syntax
```c
#include <curses.h>
int halfdelay(int tenths);
```

#### Description
The `halfdelay` subroutine sets the input mode for the current window to Half-Delay Mode and specifies tenths of seconds as the half-delay interval. The `tenths` argument must be in a range from 1 up to and including 255.

#### Flag
- `x` Instructs `wgetch` to wait `x` tenths of a second for input before timing out.

#### Parameters
- `tenths`

#### Return Values
Upon successful completion, the `halfdelay` subroutine returns OK. Otherwise, it returns ERR.

#### Related Information
The `cbreak` subroutine.

---

### has_colors Subroutine

#### Purpose
Determines whether a terminal supports color.

#### Library
Curses Library (*libcurses.a*)
Syntax
#include <curses.h>
has_colors();

Description
The `has_colors` subroutine determines whether a terminal supports color. If the terminal supports color, the `has_colors` subroutine returns TRUE. Otherwise, it returns FALSE. Because this subroutine tests for color, you can call it before the `start_color` subroutine.

The `has_colors` routine makes writing terminal-independent programs easier because you can use the subroutine to determine whether to use color or another video attribute.

Use the `can_change_colors` subroutine to determine whether a terminal that supports colors also supports changing its color definitions.

Examples
To determine whether or not a terminal supports color, use:
has_colors();

Related Information
[Curses Overview for Programming](AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

[List of Curses Subroutines](AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

[Manipulating Video Attributes](AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

**has_ic and has_il Subroutine**

**Purpose**
Query functions for terminal insert and delete capability.

**Library**
Curses Library (libcurses.a)

**Syntax**
#include <curses.h>

bool has_ic(void);
bool has_il(void);

**Description**
The `has_ic` subroutine indicates whether the terminal has insert- and delete-character capabilities.

The `has_il` subroutine indicates whether the terminal has insert- and delete-line capabilities, or can simulate them using scrolling regions.
Return Values

The has_ic subroutine returns a value of TRUE if the terminal has insert- and delete-character capabilities. Otherwise, it returns FALSE.

The has_il subroutine returns a value of TRUE if the terminal has insert- and delete-line capabilities. Otherwise, it returns FALSE.

Examples

For the has_ic subroutine:

To determine the insert capability of a terminal by returning TRUE or FALSE into the user-defined variable insert_cap, enter:

```c
int insert_cap;
insert_cap = has_ic();
```

For the has_il subroutine:

To determine the insert capability of a terminal by returning TRUE or FALSE into the user-defined variable insert_line, enter:

```c
int insert_line;
insert_line = has_il();
```

Related Information

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Understanding Terminals with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

has_il Subroutine

Purpose
Determines whether the terminal has insert-line capability.

Library
Curses Library (libcurses.a)

Syntax

```c
#include <curses.h>
has_il( )
```

Description
The has_il subroutine determines whether a terminal has insert-line capability.

Return Values
The has_il subroutine returns TRUE if terminal has insert-line capability and FALSE, if not.
Examples
To determine the insert capability of a terminal by returning TRUE or FALSE into the user-defined variable insert_line, enter:

```c
int insert_line;
insert_line = has_il();
```

Related Information
- [Curses Overview for Programming](#) in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
- [List of Curses Subroutines](#) in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
- [Understanding Terminals with Curses](#) in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

**Idlok Subroutine**

**Purpose**
Allows curses to use the hardware insert/delete line feature.

**Library**
Curses Library (libcurses.a)

**Syntax**

```c
#include <curses.h>

idlok(Window, Flag)

WINDOW *Window;
bool Flag;
```

**Description**
The *idlok* subroutine enables curses to use the hardware insert/delete line feature for terminals so equipped. If this feature is disabled, curses cannot use it. The insert/delete line feature is always considered. Enable this option only if your application needs the insert/delete line feature; for example, for a screen editor. If the insert/delete line feature cannot be used, curses will redraw the changed portions of all lines that do not match the desired line.

**Parameters**
- *Flag* Specifies whether to enable curses to use the hardware insert/delete line feature (True) or not (False).
- *Window* Specifies the window it will affect.

**Examples**
1. To enable curses to use the hardware insert/delete line feature in stdscr, enter:
   ```c
   idlok(stdscr, TRUE);
   ```
2. To force curses not to use the hardware insert/delete line feature in the user-defined window my_window, enter:
   ```c
   idlok(my_window, FALSE);
   ```
Related Information

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Setting Video Attributes and Curses Options in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

inch, mvinch, mvwinch, or winch Subroutine

Purpose
Inputs a single-byte character and rendition from a window.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

chtype inch(void);
chtype mvinch(int y, int x);

chtype mvwinch(WINDOW *win, int y, int x);
chtype winch(WINDOW *win);

Description
The inch, winch, mvinch, and mvwinch subroutines return the character and rendition, of type chtype, at the current or specified position in the current or specified window.

Parameters

*win  Specifies the window from which to get the character.

x

y

Return Values
Upon successful completion, these subroutines return the specified character and rendition. Otherwise, they return (chtype) ERR.

Examples
1. To get the character at the current cursor location in the stdscr, enter:
   ctype character;
   character = inch();

2. To get the character at the current cursor location in the user-defined window my_window, enter:
WINDOW *my_window;
chtype character;

character = winch(my_window);

3. To move the cursor to the coordinates $y = 0$, $x = 5$ and then get that character, enter:

```
chtype character;
character = mvinch(0, 5);
```

4. To move the cursor to the coordinates $y = 0$, $x = 5$ in the user-defined window `my_window` and then get that character, enter:

```
WINDOW *my_window;
chtype character;
character = mvwinch(my_window, 0, 5);
```

Related Information

- [Curses Overview for Programming](#) in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
- [List of Curses Subroutines](#) in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
- [Manipulating Characters with Curses](#) in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

### init_color Subroutine

#### Purpose
Changes a color definition.

#### Library
Curses Library (`libcurses.a`)

#### Syntax

```
#include <curses.h>

int init_color(Color, R, G, B)
register short Color, R, G, B;
```

#### Description

The `init_color` subroutine changes a color definition. A single color is defined by the combination of its red, green, and blue components. The `init_color` subroutine changes all the occurrences of the color on the screen immediately. If the color is changed successfully, this subroutine returns OK. Otherwise, it returns ERR.

**Note:** The values for the red, green, and blue components must be between 0 (no component) and 1000 (maximum amount of component). The `init_color` subroutine sets values less than 0 to 0 and values greater than 1000 to 1000.

To determine if you can change a terminal's color definitions, see the `can_change_color` subroutine.
Return Values

OK Indicates the color was changed successfully.
ERR Indicates the color was not changed.

Parameters

Color Identifies the color to change. The value of the parameter must be between 0 and COLORS-1.
R Specifies the desired intensity of the red component.
G Specifies the desired intensity of the green component.
B Specifies the desired intensity of the blue component.

Examples

To initialize the color definition for color 11 to violet on a terminal that supports at least 12 colors, use:
init_color(11,500,0,500);

Related Information

The start_color ["start_color Subroutine on page 686"] subroutine.

init_pair Subroutine

Purpose
Changes a color-pair definition.

Library
Curses Library (libcurses.a)

Syntax

```c
#include <curses.h>

init_pair(Pair, F, B)
register short Pair, F, B;
```

Description

The init_pair subroutine changes a color-pair definition. A color pair is a combination of a foreground and a background color. If you specify a color pair that was previously initialized, curses refreshes the screen and changes all occurrences of that color pair to the new definition. You must call the start_color subroutine before you call this subroutine.

Return Values

OK Indicates successful completion.
ERR Indicates the subroutine failed.
Parameters

Pair Identifies the color-pair number. The value of the Pair parameter must be between 1 and COLORS_PAIRS-1.
F Specifies the foreground color number. This number must be between 0 and COLORS-1.
B Specifies the background color number. This number must be between 0 and COLORS-1.

Examples

To initialize the color definition for color-pair 2 to a black foreground (color 0) with a cyan background (color 3), use:

init_pair(2, COLOR_BLACK, COLOR_CYAN);

Related Information

The init_color subroutine, start_color subroutine.

Curses Overview for Programming List of Curses Subroutines Manipulating Video Attributes in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

initscr and newterm Subroutine

Purpose

Initializes curses and its data structures.

Library

Curses Library (libcurses.a)

Syntax

#include <curses.h>
WINDOW *initscr(void);
SCREEN *newterm(char *type,
FILE *outfile,
FILE *infile);

Description

The initscr subroutine determines the terminal type and initializes all implementation data structures. The TERM environment variable specifies the terminal type. The initscr subroutine also causes the first refresh operation to clear the screen. If errors occur, initscr writes an appropriate error message to standard error and exits. The only subroutines that can be called before initscr or newterm are the filter, ripoffline, slk_init, use_env, and the subroutines whose prototypes are defined in <term.h>. Portable applications must not call initscr twice.

The newterm subroutine can be called as many times as desired to attach a terminal device. The type argument points to a string specifying the terminal type, except that, if type is a null pointer, the TERM environment variable is used. The outfile and infile arguments are file pointers for output to the terminal and input from the terminal, respectively. It is unspecified whether Curses modifies the buffering mode of these file pointers. The newterm subroutine should be called once for each terminal.

The initscr subroutine is equivalent to:

newterm(gentenv("TERM"), stdout, stdin); return stdscr;
If the current disposition for the signals SIGINT, SIGQUIT or SIGTSTP is SIGDFL, then the `initscr` subroutine may also install a handler for the signal, which may remain in effect for the life of the process or until the process changes the disposition of the signal.

The `initscr` and `newterm` subroutines initialise the `cur_term` external variable.

**Return Values**

Upon successful completion, the `initscr` subroutine returns a pointer to `stdscr`. Otherwise, it does not return.

Upon successful completion, the `newterm` subroutine returns a pointer to the specified terminal. Otherwise, it returns a null pointer.

**Example**

To initialize curses so that other curses subroutines can be called, use:

```c
initscr();
```

**Related Information**

The `doupdate` subroutine, `del_curterm` subroutine, `filter` subroutine, `slk_attroff` subroutine, `setupterm` subroutine.

Curses Overview for Programming, Initializing Curses, List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

### `insch`, `mvinsch`, `mvwinsch`, or `winsch` Subroutine

**Purpose**

Inserts a single-byte character and rendition in a window.

**Library**

Curses Library (libcurses.a)

**Syntax**

```c
#include <curses.h>
int insch(chtype ch);
int mvinsch(int y, chtype h);

int mvwinsch(WINDOW *win, int x, int y, chtype h);
int winsch(WINDOW *win, chtype h);
```
Description
These subroutines insert the character and rendition into the current or specified window at the current or specified position.

These subroutines do not perform wrapping or advance the cursor position. These functions perform special-character processing, with the exception that if a newline is inserted into the last line of a window and scrolling is not enabled, the behavior is unspecified.

Parameters

\[ ch \]
\[ y \]
\[ x \]
\[ *win \] Specifies the window in which to insert the character.

Return Values
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

Examples
1. To insert the character \( x \) in the stdscr, enter:
   \[
   \text{chtype } x; \text{ insch}(x); \]
2. To insert the character \( x \) into the user-defined window \( \text{my_window} \), enter:
   \[
   \text{WINDOW } *\text{my_window} \text{ ctype } x; \text{ winsch(\text{my_window}, x);} \]
3. To move the logical cursor to the coordinates \( Y=10, X=5 \) prior to inserting the character \( x \) in the stdscr, enter:
   \[
   \text{chtype } x; \text{ mvinsch}(10, 5, x); \]
4. To move the logical cursor to the coordinates \( y=10, X=5 \) prior to inserting the character \( x \) in the user-defined window \( \text{my_window} \), enter:
   \[
   \text{WINDOW } *\text{my_window}; \text{ ctype } x; \text{ mwinsch(\text{my_window}, 10, 5, x);} \]

Related Information
- [Curses Overview for Programming](AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs).
- [List of Curses Subroutines](AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs).
- [Manipulating Characters with Curses](AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs).

**insertln or wininsertln Subroutine**

Purpose
Inserts a blank line above the current line in a window.
Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
int insertln(void)

int winsertln(WINDOW *win);

Description
The insertln and winsertln subroutines insert a blank line before the current line in the current or specified window. The bottom line is no longer displayed. The cursor position does not change.

Parameters
*win Specifies the window in which to insert the blank line.

Return Values
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

Examples
1. To insert a blank line above the current line in the stdscr, enter:
   insertln();
2. To insert a blank line above the current line in the user-defined window my_window, enter:
   WINDOW *mywindow;
   winsertln(my_window);

Related Information
Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
Manipulating Characters with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

intrflush Subroutine

Purpose
Enables or disables flush on interrupt.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
int intrflush(WINDOW * win, bool bf);

Description
The intrflush subroutine specifies whether pressing an interrupt key (interrupt, suspend, or quit) will flush the input buffer associated with the current screen. If the value of $bf$ is TRUE, then flushing of the output buffer associated with the current screen will occur when an interrupt key (interrupt, suspend, or quit) is pressed. If the value of $bf$ is FALSE then no flushing of the buffer will occur when an interrupt key is pressed. The default for the option is inherited from the display driver settings. The $win$ argument is ignored.

Parameters
$bf$
$*win$ Specifies the window for which to enable or disable queue flushing.

Return Values
Upon successful completion, the intrflush subroutine returns OK. Otherwise, it returns ERR.

Examples
1. To enable queue flushing in the user-defined window $my_window$, enter:
   intrflush($my_window$, TRUE);
2. To disable queue flushing in the user-defined window $my_window$, enter:
   intrflush($my_window$, FALSE);

Related Information
List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Setting Video Attributes and Curses Options in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

keyname, key_name Subroutine

Purpose
Gets the name of keys.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
char *keyname(int c);
char *key_name(wchar_t c);
Description
The `keyname` and `key_name` subroutines generate a character string whose value describes the key \(c\). The \(c\) argument of `keyname` can be an 8-bit character or a key code. The \(c\) argument of `key_name` must be a wide character.

The string has a format according to the first applicable row in the following table:

<table>
<thead>
<tr>
<th>Input</th>
<th>Format of Returned String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible character</td>
<td>The same character</td>
</tr>
<tr>
<td>Control character</td>
<td>^X</td>
</tr>
<tr>
<td>Meta-character (keyname only)</td>
<td>M-X</td>
</tr>
<tr>
<td>Key value defined in <code>&lt;curses.h&gt;</code> (keyname only)</td>
<td>KEY_name</td>
</tr>
<tr>
<td>None of the above</td>
<td>UNKNOWN KEY</td>
</tr>
</tbody>
</table>

The meta-character notation shown above is used only, if meta-characters are enabled.

Parameter
\(c\)

Return Values
Upon successful completion, the `keyname` subroutine returns a pointer to a string as described above, Otherwise, it returns a null pointer.

Examples
```c
int key;
char *name;
keypad(stdscr, TRUE);
addstr("Hit a key");
key=getch();
name=keyname(key);
```

Note: If the Page Up key is pressed, keyname will return `KEY_PPAGE`.

Related Information
The `meta` ("meta Subroutine" on page 638) and `wgetch` ("getch, mvgetch, mvwgetch, or wgetch Subroutine" on page 611) subroutines.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

keypad Subroutine

Purpose
Enables or disables abbreviation of function keys.

Library
Curses Library (libcurses.a)

Syntax
```c
#include <curses.h>
```

632 Technical Reference, Volume 2: Base Operating System and Extensions
int keypad(WINDOW *win,
bool bf);

Description

The keypad subroutine controls keypad translation. If \( bf \) is TRUE, keypad translation is turned on. If \( bf \) is FALSE, keypad translation is turned off. The initial state is FALSE.

This subroutine affects the behavior of any function that provides keyboard input.

If the terminal in use requires a command to enable it to transmit distinctive codes when a function key is pressed, then after keypad translation is first enabled, the implementation transmits this command to the terminal before an affected input function tries to read any characters from that terminal.

Parameters

\( bf \)
\( *win \)  
Specifies the window in which to enable or disable the keypad.

Return Values

Upon successful completion, the keypad subroutine returns OK. Otherwise, it returns ERR.

Examples

To turn on the keypad in the user-defined window my_window, use:

```c
WINDOW *my_window;
keypad(my_window, TRUE);
```

Related Information

The `getch` subroutine.

The `terminfo` file format.

---

**killchar or killwchar Subroutine**

**Purpose**

Terminal environment query functions.

**Library**

Curses Library (`libcurses.a`)

**Syntax**

```c
#include <curses.h>
char killchar(void);
int killwchar(wchar_t *ch);
```

**Description**

The `killchar` subroutine returns the current line.
The `killchar` subroutine stores the current line kill character in the object pointed to by `ch`. If no line kill character has been defined, the subroutine will fail and the object pointed to by `ch` will not be changed.

**Parameters**

`*ch`

**Return Values**

The `killchar` subroutine returns the line kill character. The return value is unspecified when this character is a multi-byte character.

Upon successful completion, the `killchar` subroutine returns OK. Otherwise, it returns ERR.

**Related Information**

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

### lazySetErrorHandler Subroutine

**Purpose**

Installs an error handler into the lazy loading runtime system for the current process.

**Library**

Curses Library (libcurses.a)

**Syntax**

```c
#include <sys/ldr.h>
#include <sys/errno.h>

typedef void *handler_t
char *module;
char *symbol;
unsigned int errval;
handler_t *lazySetErrorHandler
handler_t *err_handler;
```

**Description**

This function allows a process to install a custom error handler to be called when a lazy loading reference fails to find the required module or function. This function should only be used when the main program or one of its dependent modules was linked with the -blazy option. To call `lazySetErrorHandler` from a module that is not linked with the -blazy option, you must use the -lrtl option. If you use -blazy, you do not need to specify -lrtl.

This function is not thread safe. The calling program should ensure that `lazySetErrorHandler` is not called by multiple threads at the same time.

The user-supplied error handler may print its own error message, provide a substitute function to be used in place of the called function, or call `longjmp` subroutine. To provide a substitute function that will be called instead of the originally referenced function, the error handler should return a pointer.
Parameters

Column Specifies the horizontal position to move the logical cursor to before getting the character.
Line Specifies the vertical position to move the logical cursor to before getting the character.
Window Identifies the window to get the character from and echo it into.

Return Values

Upon completion, the character code for the data key or one of the following values is returned:

**KEY_xxxx** The keypad subroutine is set to TRUE and a control key was recognized. See the curses.h file for a complete list of the key codes that can be returned.

Examples

1. To get a character and echo it to the stdscr, use:
   
   ```c
   mvgetch();
   ```

2. To get a character and echo it into stdscr at the coordinates y=20, x=30, use:
   
   ```c
   mvgetch(20, 30);
   ```

3. To get a character and echo it into the user-defined window my_window at coordinates y=20, x=30, use:
   
   ```c
   WINDOW *my_window;
   mvwgetch(my_window, 20, 30);
   ```

Related Information

The keypad subroutine, meta subroutine, nodelay subroutine, echo or noecho subroutine, notimeout subroutine, ebreak or nocbreak subroutine.

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Manipulating Characters with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

leaveok Subroutine

**Purpose**

Controls physical cursor placement after a call to the refresh subroutine.

**Library**

Curses Library (libcurses.a)

**Syntax**

```c
#include <curses.h>
```
leaveok(Window, Flag)
WINDOW *Window;
bool Flag;

Description
The leaveok subroutine controls cursor placement after a call to the refresh subroutine. If the Flag parameter is set to FALSE, curses leaves the physical cursor in the same location as logical cursor when the window is refreshed.

If the Flag parameter is set to TRUE, curses leaves the cursor as is and does not move the physical cursor when the window is refreshed. This option is useful for applications that do not use the cursor, because it reduces physical cursor motions.

By default leaveok is FALSE, and the physical cursor is moved to the same position as the logical cursor after a refresh.

Parameters
Flag Specifies whether to leave the physical cursor alone after a refresh (TRUE) or to move the physical cursor to the logical cursor after a refresh (FALSE).
Window Identifies the window to set the Flag parameter for.

Return Values
OK Indicates the subroutine completed. The leaveok subroutine always returns this value.

Examples
1. To move the physical cursor to the same location as the logical cursor after refreshing the user-defined window my_window, enter:
   WINDOW *my_window;
   leaveok(my_window, FALSE);
2. To leave the physical cursor alone after refreshing the user-defined window my_window, enter:
   WINDOW *my_window;
   leaveok(my_window, TRUE);

Related Information
The refresh subroutine. The refresh or wrefresh Subroutine on page 658 subroutine.

longname Subroutine
Purpose
Returns the verbose name of a terminal.

Library
Curses Library (libcurses.a)
Syntax

```c
#include <curses.h>
char *longname(void);
```

Description

The `longname` subroutine generates a verbose description for the current terminal. The maximum length of a verbose description is 128 bytes. It is defined only after the call to the `initscr` or `newterm` subroutines.

The area is overwritten by each call to the `newterm` subroutine, so the value should be saved if you plan on using the `longname` subroutine with multiple terminals.

Return Values

Upon successful completion, the `longname` subroutine returns a pointer to the description specified above. Otherwise, it returns a null pointer on error.

Related Information

The `initscr` ("initscr and newterm Subroutine" on page 627) subroutine, `newterm` ("newterm Subroutine" on page 644) subroutine, `setupterm` ("setupterm Subroutine" on page 674) subroutine.

**makenew Subroutine**

Purpose

Creates a new window buffer and returns a pointer.

Library

Curses Library (`libcurses.a`)

Syntax

```c
#include <curses.h>
WINDOW *makenew( )
```

Description

The `makenew` subroutine creates a new window buffer and returns a pointer to it. The `makenew` subroutine is called by the `newwin` subroutine to create the window structure. The `makenew` subroutine should not be called directly by a program.

Related Information

[Curses Overview for Programming](AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs).

[List of Curses Subroutines](AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs).

[Understanding Terminals with Curses](AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs).
meta Subroutine

Purpose
Enables/disables meta-keys.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

int meta(WINDOW *win,
bool bf);

Description
Initially, whether the terminal returns 7 or 8 significant bits on input depends on the control mode of the display driver. To force 8 bits to be returned, invoke the meta subroutine (win, TRUE). To force 7 bits to be returned, invoke the meta subroutine (win, FALSE). The win argument is always ignored.

If the terminfo capabilities smm (meta_on) and rmm (meta_off) are defined for the terminal, smm is sent to the terminal when meta (win, TRUE) is called and rmm is sent when meta (win, FALSE) is called.

Parameters

bf
"win"

Return Values
Upon successful completion, the meta subroutine returns OK. Otherwise, it returns ERR.

Examples
1. To request an 8-bit character return when using a getch routine, enter:
   WINDOW *some_window;
   meta(some_window, TRUE);
2. To strip the highest bit off the character returns in the user-defined window my_window, enter:
   WINDOW *some_window;
   meta(some_window, FALSE);

Related Information
The getch ("getch, mvgetch, mvwgetch, or wgetch Subroutine" on page 611) subroutine.

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Manipulating Characters with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
move or wmove Subroutine

Purpose
Window location cursor functions.

Library
Curses Library (libcurses.a)

Syntax
int ( x);

int wmove (WINDOW *win,
int y,
int x);

Description
The move and wmove subroutines move the logical cursor associated with the current or specified window to (y, x) relative to the window's origin. This subroutine does not move the cursor of the terminal until the next refresh ("refresh or wrefresh Subroutine" on page 658) operation.

Parameters
y
x
"win

Return Values
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

Examples
1. To move the logical cursor in the stdscr to the coordinates y = 5, x = 10, use:
   move(5, 10);
2. To move the logical cursor in the user-defined window my_window to the coordinates y = 5, x = 10, use:
   WINDOW *my_window;
   wmove(my_window, 5, 10);

Related Information
The getch ("getch, mvgetch, mvwgetch, or wgetch Subroutine" on page 611) and refresh ("refresh or wrefresh Subroutine" on page 658) subroutines.

Controlling the Cursor with Curses, Curses Overview for Programming, List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

mvcur Subroutine

Purpose
Output cursor movement commands to the terminal.
Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

int mvcur(int oldrow,
int oldcol,
int newrow,
int newcol);

Description
The `mvcur` subroutine outputs one or more commands to the terminal that move the terminal's cursor to
(newrow, newcol), an absolute position on the terminal screen. The (oldrow, oldcol) arguments specify the
former cursor position. Specifying the former position is necessary on terminals that do not provide
coordinate-based movement commands. On terminals that provide these commands, Curses may select a
more efficient way to move the cursor based on the former position. If (newrow, newcol) is not a valid
address for the terminal in use, the `mvcur` subroutine fails. If (oldrow, oldcol) is the same as (newrow,
newcol), `mvcur` succeeds without taking any action. If `mvcur` outputs a cursor movement command, it
updates its information concerning the location of the cursor on the terminal.

Parameters
newcol
newrow
oldrow
oldcol

Return Values
Upon successful completion, the `mvcur` subroutine returns OK. Otherwise, it returns ERR.

Examples
1. To move the physical cursor from the coordinates y = 5, x = 15 to y = 25, x = 30, use:
   `mvcur(5, 15, 25, 30);`
2. To move the physical cursor from unknown coordinates to y = 5, x = 0, use:
   `mvcur(50, 50, 5, 0);`

In this example, the physical cursor's current coordinates are unknown. Therefore, arbitrary values are
assigned to the OldLine and OldColumn parameters and the desired coordinates are assigned to the
NewLine and NewColumn parameters. This is called an absolute move.

Related Information
The `doupdate` subroutine, `is_linetouched` subroutine, `move` subroutine, `refresh` subroutine,
`refresh or wrefresh Subroutine` on page 658 subroutine.
mvwin Subroutine

Purpose
Moves a window or subwindow to the specified coordinates.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

int mvwin
(WINDOW *win,
 int y,
 int x);

Description
The mvwin subroutine moves the specified window so that its origin is at position (y, x). If the move causes any portion of the window to extend past any edge of the screen, the function fails and the window is not moved.

Parameters

*win
x
y

Return Values
Upon successful completion, the mvwin subroutine returns OK. Otherwise, it returns ERR.

Examples
1. To move the user-defined window my_window from its present location to the upper left corner of the terminal, enter:
   WINDOW *my_window;
   mvwin(my_window, 0, 0);
2. To move the user-defined window my_window from its present location to the coordinates y = 20, x = 10, enter:
   WINDOW *my_window;
   mvwin(my_window, 20, 10);

Related Information
The derwin ("derwin, newwin, or subwin Subroutine" on page 646) subroutine, doupdate ("doupdate, refresh, wnoutrefresh, or wrefresh Subroutines" on page 707) subroutine, is_linetouched ("is_linetouched, is_wintouched, touchline, touchwin, untouchwin, or wtouchin Subroutine" on page 698) subroutine.

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
newpad, pnoutrefresh, prefresh, or subpad Subroutine

Purpose
Pad management functions.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

WINDOW *newpad
(int nlines,
int ncols);

int pnoutrefresh
(WINDOW *pad,
int pminrow,
int pmincol,
int sminrow,
int smincol,
int smaxrow,
int smaxcol);

int prefresh
(WINDOW *pad,
int pminrow,
int pmincol,
int sminrow,
int smincol,
int smaxrow,
int smaxcol);

WINDOW *
subpad
(WINDOW *orig,
int nlines,
int ncols,
int begin_y,
int begin_x);

Description
The newpad subroutine creates a specialised WINDOW data structure with nlines lines and ncols columns. A pad is similar to a window, except that it is not associated with a viewable part of the screen. Automatic refreshes of pads do not occur.

The subpad subroutine creates a subwindow within a pad with nlines lines and ncols columns. Unlike the subwin subroutine, which uses screen coordinates, the window is at a position (begin_y, begin_x) on the pad. The window is made in the middle of the window orig, so that changes made to one window affects both windows.

The prefresh (“prefresh or pnoutrefresh Subroutine” on page 653) or pnoutrefresh (“prefresh or pnoutrefresh Subroutine” on page 653) subroutines are analogous to the wrefresh and wnoutrefresh subroutines except that they relate to pads instead of windows. The additional arguments indicate what
part of the pad and screen are involved. The \textit{pminrow} and \textit{pmincol} arguments specify the origin of the rectangle to be displayed in the screen. The lower right-hand corner of the rectangle to be displayed in the pad is calculated from the screen coordinates, since the rectangles must be the same size. Both rectangles must be entirely contained within their respective structures. Negative values of \textit{pminrow}, \textit{pmincol}, \textit{sminrow} or \textit{smincol} are treated as if they were zero.

\textbf{Parameters}

\begin{itemize}
  \item \texttt{ncols}
  \item \texttt{nlines}
  \item \texttt{begin\_x}
  \item \texttt{begin\_y}
  \item \texttt{*orig}
  \item \texttt{*pad}
  \item \texttt{pminrow}
  \item \texttt{pmincol}
  \item \texttt{sminrow}
  \item \texttt{smincol}
  \item \texttt{smaxrow}
  \item \texttt{smaxcol}
\end{itemize}

\textbf{Return Values}

Upon successful completion, the \texttt{newpad} and \texttt{subpad} subroutines return a pointer to the pad structure. Otherwise, they return a null pointer.

Upon successful completion, the \texttt{pnoutrefresh} and \texttt{prefresh} subroutines return OK. Otherwise, they return ERR.

\textbf{Examples}

For the \texttt{newpad} subroutine:
1. To create a new pad and save the pointer to it in \texttt{my\_pad}, enter:
   
   \begin{verbatim}
   WINDOW *my_pad;
   my_pad = newpad(5, 10);
   my_pad is now a pad 5 lines deep, 10 columns wide.
   \end{verbatim}
2. To create a pad and save the pointer to it in \texttt{my\_pad}, which is flush with the right side of the terminal, enter:
   
   \begin{verbatim}
   WINDOW *my_pad;
   my_pad = newpad(5, 0);
   my_pad is now a pad 5 lines deep, extending to the far right side of the terminal.
   \end{verbatim}
3. To create a pad and save the pointer to it in \texttt{my\_pad}, which fills the entire terminal, enter:
   
   \begin{verbatim}
   WINDOW *my_pad;
   my_pad = newpad(0, 0);
   my_pad is now a pad that fills the entire terminal.
   \end{verbatim}
4. To create a very large pad and display part of it on the screen, enter:
   
   \begin{verbatim}
   WINDOW *my_pad;
   my_pal = newpad(120,120);
   prefresh (my_pal, 0,0,0,0,20,30);
   \end{verbatim}
This causes the first 21 rows and first 31 columns of the pad to be displayed on the screen. The upper left coordinates of the resulting rectangle are (0,0) and the bottom right coordinates are (20,30).

For the `prefresh` or `pnoutrefresh` subroutines:
1. To update the user-defined `my_pad` pad from the upper-left corner of the pad on the terminal with the upper-left corner at the coordinates `Y=20, X=10` and the lower-right corner at the coordinates `Y=30, X=25` enter:
   ```c
   WINDOW *my_pad;
prefresh(my_pad, 0, 0, 20, 10, 30, 25);
   ```
2. To update the user-defined `my_pad1` and `my_pad2` pads and output them both to the terminal in one burst of output, enter:
   ```c
   WINDOW *my_pad1, *my_pad2;
pnoutrefresh(my_pad1, 0, 0, 20, 10, 30, 25);
pnoutrefresh(my_pad2, 0, 0, 0, 0, 10, 5);
doupdate();
   ```

For the `subpad` subroutine:

To create a subpad, use:
   ```c
   WINDOW *orig, *mypad;
   orig = newpad(100, 200);
mypad = subpad(orig, 30, 5, 25, 180);
   ```

The parent pad is 100 lines by 200 columns. The subpad is 30 lines by 5 columns and starts in line 25, column 180 of the parent pad.

**Related Information**
The `derwin` ("derwin, newwin, or subwin Subroutine" on page 646) subroutine, `doupdate` ("doupdate, refresh, wnoutrefresh, or wrefresh Subroutines" on page 707) subroutine, `is_linetouched` ("is_linetouched, isointouched, touchline, touchwin, untouchwin, or wtouchin Subroutine" on page 698) subroutine, `prefresh` ("prefresh or pnoutrefresh Subroutine" on page 653) or `pnoutrefresh` ("prefresh or pnoutrefresh Subroutine" on page 653) subroutine, and `subpad` ("subpad Subroutine" on page 687) subroutine.

For details on `newterm`, see "newterm Subroutine".

**newterm Subroutine**

**Purpose**
Initializes curses and its data structures for a specified terminal.

**Library**
Curses Library (libcurses.a)

**Syntax**
```c
#include <curses.h>
```
**Description**

The `newterm` subroutine initializes curses and its data structures for a specified terminal. Use this subroutine instead of the `initscr` subroutine if you are writing a program that sends output to more than one terminal. You should also use this subroutine if your program requires indication of error conditions so that it can run in a line-oriented mode on terminals that do not support a screen-oriented program.

If you are directing your program’s output to more than one terminal, you must call the `newterm` subroutine once for each terminal. You must also call the `endwin` subroutine for each terminal to stop curses and restore the terminal to its previous state.

**Parameters**

- **InFile** Identifies the input device file.
- **OutFile** Identifies the output device file.
- **Type** Specifies the type of output terminal. This parameter is the same as the `$TERM` environment variable for that terminal.

**Return Values**

The `newterm` subroutine returns a variable of type `SCREEN *`. You should save this reference to the terminal within your program.

**Examples**

1. To initialize curses on a terminal represented by the lft device file as both the input and output terminal, open the device file with the following:
   ```c
   fdfile = fopen("/dev/lft0", "r+");
   ```
   Then, use the `newterm` subroutine to initialize curses on the terminal and save the new terminal in the `my_terminal` variable as follows:
   ```c
   char termname [] = "terminaltype";
   SCREEN *my_terminal;
   my_terminal = newterm(termname, fdfile, fdfile);
   ```

2. To open the device file `/dev/lft0` as the input terminal and the `/dev/tty0` (an ibm3151) as the output terminal, do the following:
   ```c
   fdifile = fopen("/dev/lft0", "r");
   fdofile = fopen("/dev/tty0", "w");
   ```
   ```c
   SCREEN *my_terminal2;
   my_terminal2 = newterm("ibm3151", fdofile, fdifile);
   ```

3. To use stdin for input and stdout for output, do the following:
   ```c
   char termname [] = "terminaltype";
   SCREEN *my_terminal;
   my_terminal = newterm(termname, stdout, stdin);
   ```

**Related Information**

The `endwin` subroutine, `initscr` subroutine.
derwin, newwin, or subwin Subroutine

Purpose
Window creation subroutines.

Library
Curses Library (libcurses.a)

Syntax
```
#include <curses.h>

WINDOW *derwin(WINDOW *orig,
    int nlines,
    int ncols,
    int begin_y,
    int begin_x);

WINDOW *newwin(int nlines,
    int ncols,
    int begin_y,
    int begin_x);

WINDOW *subwin(WINDOW *orig,
    int nlines,
    int ncols,
    int begin_y,
    int begin_x);
```

Description
The `derwin` subroutine is the same as the `subwin` subroutine except that `begin_y` and `begin_x` are relative to the origin of the window `orig` rather than absolute screen positions.

The `newwin` subroutine creates a new window with `nlines` lines and `ncols` columns, positioned so that the origin is at `(begin_y, begin_x)`. If `nlines` is zero, it defaults to `LINES - begin_y`; if `ncols` is zero, it defaults to `COLS - begin_x`.

The `subwin` subroutine creates a new window with `nlines` lines and `ncols` columns, positioned so that the origin is at `(begin_y, begin_x)`. (This position is an absolute screen position, not a position relative to the window `orig`.) If any part of the new window is outside `orig`, the subroutine fails and the window is not created.

Parameters
- `ncols`
- `nlines`
- `begin_y`
- `begin_x`

Return Values
Upon successful completion, these subroutines return a pointer to the new window. Otherwise, they return a null pointer.
Examples

For the `derwin` and `newwin` subroutines:

1. To create a new window, enter:
   ```c
   WINDOW *my_window;
   my_window = newwin(5, 10, 20, 30);
   ```
   
   `my_window` is now a window 5 lines deep, 10 columns wide, starting at the coordinates y = 20, x = 30. That is, the upper left corner is at coordinates y = 20, x = 30, and the lower right corner is at coordinates y = 24, x = 39.

2. To create a window that is flush with the right side of the terminal, enter:
   ```c
   WINDOW *my_window;
   my_window = newwin(5, 0, 20, 30);
   ```
   
   `my_window` is now a window 5 lines deep, extending all the way to the right side of the terminal, starting at the coordinates y = 20, x = 30. The upper left corner is at coordinates y = 20, x = 30, and the lower right corner is at coordinates y = 24, x = lastcolumn.

3. To create a window that fills the entire terminal, enter:
   ```c
   WINDOW *my_window;
   my_window = newwin(0, 0, 0, 0);
   ```
   
   `my_window` is now a screen that is a window that fills the entire terminal's display.

For the `subwin` subroutine:

1. To create a subwindow, use:
   ```c
   WINDOW *my_window, *my_sub_window;
   my_window = newwin("derwin, newwin, or subwin Subroutine" on page 646, 5, 10, 20, 30);
   ```
   
   `my_sub_window` is now a subwindow 2 lines deep, 5 columns wide, starting at the same coordinates of its parent window `my_window`. That is, the subwindow's upper-left corner is at coordinates y = 20, x = 30 and lower-right corner is at coordinates y = 21, x = 34.

2. To create a subwindow that is flush with the right side of its parent, use:
   ```c
   WINDOW *my_window, *my_sub_window;
   my_window = newwin("derwin, newwin, or subwin Subroutine" on page 646, 5, 10, 20, 30);
   my_sub_window = subwin(my_window, 2, 0, 20, 30);
   ```
   
   `my_sub_window` is now a subwindow 2 lines deep, extending all the way to the right side of its parent window `my_window`, and starting at the same coordinates. That is, the subwindow's upper-left corner is at coordinates y = 20, x = 30 and lower-right corner is at coordinates y = 21, x = 39.

3. To create a subwindow in the lower-right corner of its parent, use:
   ```c
   WINDOW *my_window, *my_sub_window;
   my_window = newwin("derwin, newwin, or subwin Subroutine" on page 646, 5, 10, 20, 30);
   my_sub_window = subwin(my_window, 0, 0, 22, 35);
   ```
my_sub_window is now a subwindow that fills the bottom right corner of its parent window, my_window, starting at the coordinates y = 22, x = 35. That is, the subwindow’s upper-left corner is at coordinates y = 22, x = 35 and lower-right corner is at coordinates y = 24, x = 39.

Related Information
The endwin (endwin Subroutine on page 604), initscr (initscr and newterm Subroutine on page 627) subroutines.

nl or nonl Subroutine

Purpose
Enables/disables newline translation.

Library
Curses Library (libcurses.a)

Syntax
```c
#include <curses.h>
int nl(void);
int nonl(void);
```

Description
The nl subroutine enables a mode in which carriage return is translated to newline on input. The nonl subroutine disables the above translation. Initially, the above translation is enabled.

Return Values
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

Examples
1. To instruct wgetch to translate the carriage return into a newline, enter:
   ```
   nl();
   ```
2. To instruct wgetch not to translate the carriage return, enter:
   ```
   nonl();
   ```

Related Information
The refresh (refresh or wrefresh Subroutine on page 658) subroutine, waddch (addch, mvaddch, mvwaddch, or waddch Subroutine on page 573) subroutine.

nodelay Subroutine

Purpose
Enables or disables block during read.
Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

int nodelay(WINDOW *win, bool bf);

Description
The nodelay subroutine specifies whether Delay Mode or No Delay Mode is in effect for the screen associated with the specified window. If bf is TRUE, this screen is set to No Delay Mode. If bf is FALSE, this screen is set to Delay Mode. The initial state is FALSE.

Parameters
bf
*win

Return Values
Upon successful completion, the nodelay subroutine returns OK. Otherwise, it returns ERR.

Examples
1. To cause the wgetch subroutine to return an error message, if no input is ready in the user-defined window my_window, use:
   nodelay(my_window, TRUE);
2. To allow for a delay when retrieving a character in the user-defined window my_window, use:
   WINDOW *my_window;
   nodelay(my_window, FALSE);

Related Information
The halfdelay ("halfdelay Subroutine" on page 620) subroutine, wgetch ("getch, mvgetch, mvwgetch, or wgetch Subroutine" on page 611) subroutine.

notimeout, timeout, wtimeout Subroutine

Purpose
Controls blocking on input.

Library
Curses Library (libcurses.a)

Curses Syntax
#include <curses.h>

int notimeout
(WINDOW *win, bool bf);
void timeout
(int delay);

void wtimeout
(WINDOW *win,
int delay);

Description
The notimeout subroutine specifies whether Timeout Mode or No Timeout Mode is in effect for the screen associated with the specified window. If bf is TRUE, this screen is set to No Timeout Mode. If bf is FALSE, this screen is set to Timeout Mode. The initial state is FALSE.

The timeout and wtimeout subroutines set blocking or non-blocking read for the current or specified window based on the value of delay:

- delay < 0: One or more blocking reads (indefinite waits for input) are used.
- delay = 0: One or more non-blocking reads are used. Any Curses input subroutine will fail if every character of the requested string is not immediately available.
- delay > 0: Any Curses input subroutine blocks for delay milliseconds and fails if there is still no input.

Parameters
*win
bf

Return Values
Upon successful completion, the notimeout subroutine returns OK. Otherwise, it returns ERR.

The timeout and wtimeout subroutines do not return a value.

Examples
To set the flag so that the wgetch subroutine does not set the timer when getting characters from the my_win window, use:

WINDOW *my_win;
notimeout(my_win, TRUE);

Related Information
The wgetch ("wgetch, mvwgetch, mwwgetch, or wgetch Subroutine" on page 611), halfdelay ("halfdelay Subroutine" on page 620), nodelay ("nodelay Subroutine" on page 648), and notimeout ("notimeout, timeout, wtimeout Subroutine" on page 649) subroutines.

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Manipulating Characters with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Getting Characters in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
overlay or overwrite Subroutine

Purpose
Copies one window on top of another.

Library
Curses Library (libcurses.a)

Syntax

```c
WINDOW *dstwin);
int overwrite(const WINDOW *srcwin,
WINDOW *dstwin);
```

Description
The overlay and overwrite subroutines overlay srcwin on top of dstwin. The scrwin and dstwin arguments need not be the same size; only text where the two windows overlap is copied.

The overwrite subroutine copies characters as though a sequence of win_wch and wadd_wch subroutines were performed with the destination window’s attributes and background attributes cleared.

The overlay subroutine does the same thing, except that, whenever a character to be copied is the background character of the source window, the overlay subroutine does not copy the character but merely moves the destination cursor the width of the source background character.

If any portion of the overlaying window border is not the first column of a multi-column character then all the column positions will be replaced with the background character and rendition before the overlay is done. If the default background character is a multi-column character when this occurs, then these subroutines fail.

Parameters

srcwin
deswin

Return Values
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

Examples
1. To copy my_window on top of other_window, excluding spaces, use:
   ```c
   WINDOW *my_window, *other_window;
   overlay(my_window, other_window);
   ```
2. To copy my_window on top of other_window, including spaces, use:
   ```c
   WINDOW *my_window, *other_window;
   overwrite(my_window, other_window);
   ```

Related Information
The copywin subroutine.

Curses Overview for Programming List of Curses Subroutines Manipulating Window Data with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
pair_content Subroutine

Purpose
Returns the colors in a color pair.

Library
Curses Library (libcurses.a)

Curses Syntax
#include <curses.h>

pair_content (Pair, F, B)
short Pair;
short *F, *B;

Description
The pair_content subroutine returns the colors in a color pair. A color pair is made up of a foreground and background color. You must call the start_color subroutine before calling the pair_content subroutine.

Note: The color pair must already be initialized before calling the pair_content subroutine.

Return Values
OK Indicates the subroutine completed successfully.
ERR Indicates the pair has not been initialized.

Parameters
Pair Identifies the color-pair number. The Pair parameter must be between 1 and COLORS_PAIRS-1.
F Points to the address where the foreground color will be stored. The F parameter will be between 0 and COLORS-1.
B Points to the address where the background color will be stored. The B parameter will be between 0 and COLORS-1.

Example
To obtain the foreground and background colors for color-pair 5, use:
short *f, *b;
pair_content(5, f, b);

For this subroutine to succeed, you must have already initialized the color pair. The foreground and background colors will be stored at the locations pointed to by f and b.

Related Information
The start_color subroutine, init_pair subroutine.
Curses Overview for Programming List of Curses Subroutines Manipulating Video Attributes Working with Color in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
prefresh or pnoutrefresh Subroutine

Purpose

Updates the terminal and curscr (current screen) to reflect changes made to a pad.

Library

Curses Library (libcurses.a)

Syntax

```c
#include <curses.h>

prefresh(Pad, PY, PX, TTY, TTX, TBY, TBX)
WINDOW *Pad;
int PY, PX, TTY;
int TTX, TBY, TBX;

pnoutrefresh(Pad, PY, PX, TTY, TTX, TBY, TBX)
WINDOW *Pad;
int PY, PX, TTY;
int TTX, TBY, TBX;
```

Description

The `prefresh` and `pnoutrefresh` subroutines are similar to the `wrefresh` ("refresh or wrefresh Subroutine" on page 658) and `wnoutrefresh` ("doupdate, refresh, wnoutrefresh, or wrefresh Subroutines" on page 707) subroutines. They are different in that pads, instead of windows, are involved, and additional parameters are necessary to indicate what part of the pad and screen are involved.

The `PX` and `PY` parameters specify the upper left corner, in the pad, of the rectangle to be displayed. The `TTX`, `TTY`, `TBX`, and `TBY` parameters specify the edges, on the screen, for the rectangle to be displayed in. The lower right corner of the rectangle to be displayed is calculated from the screen coordinates, since both rectangle and pad must be the same size. Both rectangles must be entirely contained within their respective structures.

The `prefresh` subroutine copies the specified portion of the pad to the physical screen. If you wish to output several pads at once, call `pnoutrefresh` for each pad and then issue one call to `doupdate`. This updates the physical screen once.

Parameters

- **Pad** Specifies the pad to be refreshed.
- **PX** (Pad’s x-coordinate) Specifies the upper-left column coordinate, in the pad, of the rectangle to be displayed.
- **PY** (Pad’s y-coordinate) Specifies the upper-left row coordinate, in the pad, of the rectangle to be displayed.
- **TBX** (Terminal’s Bottom x-coordinate) Specifies the lower-right column coordinate, on the terminal, for the pad to be displayed in.
- **TBY** (Terminal’s Bottom y-coordinate) Specifies the lower-right row coordinate, on the terminal, for the pad to be displayed in.
- **TTX** (Terminal’s Top x-coordinate) Specifies the upper-left column coordinate, on the terminal, for the pad to be displayed in.
- **TTY** (Terminal’s Top Y coordinate) Specifies the upper-left row coordinate, on the terminal, for the pad to be displayed in.
Examples

1. To update the user-defined my_pad pad from the upper-left corner of the pad on the terminal with the upper-left corner at the coordinates Y=20, X=10 and the lower-right corner at the coordinates Y=30, X=25 enter:

   WINDOW *my_pad;
prefresh(my_pad, 0, 0, 20, 10, 30, 25);

2. To update the user-defined my_pad1 and my_pad2 pads and output them both to the terminal in one burst of output, enter:

   WINDOW *my_pad1; *my_pad2;
pnoutrefresh(my_pad1, 0, 0, 20, 10, 30, 25);
pnoutrefresh(my_pad2, 0, 0, 0, 0, 10, 5);
doupdate();

Related Information

- Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
- List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
- Manipulating Window Data with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

printw, wprintw, mvprintw, or mvwprintw Subroutine

Purpose

Performs a printf command on a window using the specified format control string.

Library

Curses Library (libcurses.a)

Syntax

```c
#include <curses.h>

printw( Format, [Argument ...])
char *Format, *Argument;

wprintw( Window, Format, [Argument ...])
WINDOW *Window;
char *Format, *Argument;

mvprintw( Line, Column, Format, [Argument ...])
int Line, Column;
char *Format, *Argument;

mvwprintw(Window, Line, Column, Format, [Argument ...])
WINDOW *Window;
int Line, Column;
char *Format, *Argument;
```

Description

The printw, wprintw, mvprintw, and mvwprintw subroutines perform output on a window by using the specified format control string. However, the waddch (*addch, mvaddch, mvwaddch, or waddch
Subroutine" on page 573 subroutine is used to output characters in a given window instead of invoking the printf subroutine. The mvprintw and mvwprintw subroutines move the logical cursor before performing the output.

Use the printw and mvprintw subroutines on the stdscr and the wprintw and mvwprintw subroutines on user-defined windows.

Note: The maximum length of the format control string after expansion is 512 bytes.

Parameters

- **Argument** Specifies the item to print. See the printf subroutine for more details.
- **Column** Specifies the horizontal position to move the cursor to before printing.
- **Format** Specifies the format for printing the Argument parameter. See the printf subroutine.
- **Line** Specifies the vertical position to move the cursor to before printing.
- **Window** Specifies the window to print into.

Examples

1. To print the user-defined integer variables x and y as decimal integers in the stdscr, enter:
   ```c
   int x, y;
   printw("%d%d", x, y);
   ```

2. To print the user-defined integer variables x and y as decimal integers in the user-defined window my_window, enter:
   ```c
   int x, y;
   WINDOW *my_window;
   wprintw(my_window, "%d%d", x, y);
   ```

3. To move the logical cursor to the coordinates y = 5, x = 10 before printing the user-defined integer variables x and y as decimal integers in the stdscr, enter:
   ```c
   int x, y;
   mvprintw(5, 10, "%d%d", x, y);
   ```

4. To move the logical cursor to the coordinates y = 5, x = 10 before printing the user-defined integer variables x and y as decimal integers in the user-defined window my_window, enter:
   ```c
   int x, y;
   WINDOW *my_window;
   mvwprintw(my_window, 5, 10, "%d%d", x, y);
   ```

Related Information

The waddch (“addch, mvaddch, mvwaddch, or waddch Subroutine” on page 573) subroutine, printf subroutine.

The printf command.

Curses Overview for Programming | List of Curses Subroutines | Manipulating Characters with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

putp, tputs Subroutine

**Purpose**

Outputs commands to the terminal.

**Library**

Curses Library (libcurses.a)
Syntax
#include <curses.h>

int putp(const char *str);

int tputs(const char *str, int affcn, int (*putfunc)(int));

Description
These subroutines output commands contained in the terminfo database to the terminal.

The putp subroutine is equivalent to tputs(str, 1, putchar). The output of the putp subroutine always goes to stdout, not to the filedes specified in the setupterm subroutine.

The tputs subroutine outputs str to the terminal. The str argument must be a terminfo string variable or the return value from the tgetstr, tgoto, tigestr, or tparm subroutines. The affcnt argument is the number of lines affected, or 1 if not applicable. If the terminfo database indicates that the terminal in use requires padding after any command in the generated string, the tputs subroutine inserts pad characters into the string that is sent to the terminal, at positions indicated by the terminfo database. The tputs subroutine outputs each character of the generated string by calling the user-supplied putfunc subroutine (see below).

The user-supplied putfunc subroutine (specified as an argument to the tputs subroutine is either putchar or some other subroutine with the same prototype. The tputs subroutine ignores the return value of the putfunc subroutine.

Parameters
*str
affcnt
*putfunc

Return Values
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

Examples
For the putp subroutine:

To call the tputs(my_string, 1, putchar) subroutine, enter:
char *my_string;
putp(my_string);

For the tputs subroutine:
1. To output the clear screen sequence using the user-defined putchar-like subroutine my_putchar, enter:
   int my_putchar();
tputs(clear_screen, 1 ,my_putchar);
2. To output the escape sequence used to move the cursor to the coordinates x=40, y=18 through the user-defined putchar-like subroutine my_putchar, enter:
   int my_putchar();
tputs(tparm(cursor_address, 18, 40), 1, my_putchar);
Related Information

The **doupdate** ("doupdate, refresh, wnoutrefresh, or wrefresh Subroutines" on page 707) subroutine, **is_linetouched** ("is_linetouched, is_wintouched, touchline, touchwin, untouchwin, or wtouchin Subroutine" on page 698) subroutine, **putchar** subroutine, **tgetent** ("tgetent, tgetflag, tgetnum, tgetstr, or tgoto Subroutine" on page 690) subroutine, **tigetflag** ("tigetflag, tigetnum, tigetstr, or tparm Subroutine" on page 694) subroutine, **tputs** ("putp, tputs Subroutine" on page 655) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.*

List of Curses Subroutines in *AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.*

Understanding Terminals with Curses in *AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.*

---

### raw or noraw Subroutine

**Purpose**

Places the terminal into or out of raw mode.

**Library**

Curses Library (*libcurses.a*)

**Syntax**

```
#include <curses.h>
raw();
noraw();
```

**Description**

The **raw** or **noraw** subroutine places the terminal into or out of raw mode, respectively. RAW mode is similar to CBREAK mode (**cbreak** or **nocbreak** ("cbreak, nocbreak, noraw, or raw Subroutine" on page 585) subroutine). In RAW mode, the system immediately passes typed characters to the user program. The interrupt, quit, and suspend characters are passed uninterrupted, instead of generating a signal. RAW mode also causes 8-bit input and output.

To get character-at-a-time input without echoing, call the **cbreak** and **noecho** subroutines. Most interactive screen-oriented programs require this sort of input.

**Return Values**

**OK**  Indicates the subroutine completed. The **raw** and **noraw** routines always return this value.

**Examples**

1. To place the terminal into raw mode, use:
   
   ```
   raw();
   ```

2. To place the terminal out of raw mode, use:

   ```
   noraw();
   ```
Related Information

The `getch` subroutine, `cbreak` or `nocbreak` subroutine, and `refresh` or `wrefresh` subroutine.

Curses Overview for Programming | List of Curses Subroutines | Understanding Terminals with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

refresh or wrefresh Subroutine

Purpose
Updates the terminal’s display and the curscr to reflect changes made to a window.

Library
Curses Library (libcurses.a)

Syntax
```
#include <curses.h>

refresh()

wrefresh(Window)
WINDOW *Window;
```

Description
The `refresh` or `wrefresh` subroutines update the terminal and the curscr to reflect changes made to a window. The `refresh` subroutine updates the stdscr. The `wrefresh` subroutine refreshes a user-defined window.

Other subroutines manipulate windows but do not update the terminal’s physical display to reflect their changes. Use the `refresh` or `wrefresh` subroutines to update a terminal’s display after internal window representations change. Both subroutines check for possible scroll errors at display time.

Note: The physical terminal cursor remains at the location of the window’s cursor during a refresh, unless the `leaveok` subroutine is enabled.

The `refresh` and `wrefresh` subroutines call two other subroutines to perform the refresh operation. First, the `wnoutrefresh` subroutine copies the designated window structure to the terminal. Then, the `doupdate` subroutine updates the terminal’s display and the cursor.

Parameters

**Window** Specifies the window to refresh.

Examples

1. To update the terminal’s display and the current screen structure to reflect changes made to the standard screen structure, use:
   ```
   refresh();
   ```

2. To update the terminal and the current screen structure to reflect changes made to a user-defined window called `my_window`, use:
To restore the terminal to its state at the last refresh, use:
```c
wrefresh(curscr);
```
This subroutine is useful if the terminal becomes garbled for any reason.

---

**Related Information**

The `doupdate` subroutine, `leaveok` subroutine, `wnoutrefresh` subroutine.

**reset_prog_mode Subroutine**

**Purpose**
Restores the terminal to program mode.

**Library**
Curses Library (`libcurses.a`)

**Syntax**
```c
#include <curses.h>
reset_prog_mode();
```

**Description**
The `reset_prog_mode` subroutine restores the terminal to program or in curses mode.

The `reset_prog_mode` subroutine is a low-level routine and normally would not be called directly by a program.

**Related Information**
AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

**reset_shell_mode Subroutine**

**Purpose**
Restores the terminal to shell mode.

**Library**
Curses Library (`libcurses.a`)
Syntax
#include <curses.h>
reset_shell_mode( )

Description
The reset_shell_mode subroutine restores the terminal into shell, or "out of curses," mode. This happens automatically when the endwin subroutine is called.

Related Information
The endwin subroutine.

resetterm Subroutine

Purpose
Resets terminal modes to what they were when the saveterm subroutine was last called.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
resetterm( )

Description
The resetterm subroutine resets terminal modes to what they were when the saveterm subroutine was last called.

The resetterm subroutine is called by the endwin subroutine, and should normally not be called directly by a program.

Related Information
Curses Overview for Programming
List of Curses Subroutines
Understanding Terminals with Curses
resetty, savetty Subroutine

Purpose
Saves/restores the terminal mode.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
int resetty(void);
int savetty(void);

Description
The resetty subroutine restores the program mode as of the most recent call to the savetty subroutine.

The savetty subroutine saves the state that would be put in place by a call to the reset_prog_mode subroutine.

Return Values
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

Examples
To restore the terminal to the state it was in at the last call to savetty, enter:
resetty();

Related Information
The def_prog_mode subroutine, endwin subroutine, savetty subroutine.

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Understanding Terminals with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

restartterm Subroutine

Purpose
Re-initializes the terminal structures after a restore.

Library
Curses Library (libcurses.a)
Syntax
#include <curses.h>
#include <term.h>

restartterm (Term, FileNumber, ErrorCode)
char *Term;
int FileNumber;
int *ErrorCode;

Description
The restartterm subroutine is similar to the setupterm subroutine except that it is called after restoring memory to a previous state. For example, you would call the restartterm subroutine after a call to scr_restore if the terminal type has changed. The restartterm subroutine assumes that the windows and the input and output options are the same as when memory was saved, but the terminal type and baud rate may be different.

Parameters
Term Specifies the terminal name to obtain the terminal for. If 0 is passed for the parameter, the value of the $TERM environment variable is used.
FileNumber Specifies the output file’s file descriptor (1 equals standard out).
ErrorCode Specifies a pointer to an integer to return the error code to. If 0, then the restartterm subroutine exits with an error message instead of returning.

Example
To restart an aixterm after a previous memory save and exit on error with a message, enter:
restartterm("aixterm", 1, (int*)0);

Prerequisite Information
Curses Overview for Programming and Understanding Terminals with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Related Information
The setupterm subroutine.

ripoffline Subroutine

Purpose
Reserves a line for a dedicated purpose.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

int ripoffline(int line,
int (*init)(WINDOW *win,
int columns));
Description

The ripoffline subroutine reserves a screen line for use by the application.

Any call to the ripoffline subroutine must precede the call to the initscr or newterm subroutine. If line is positive, one line is removed from the beginning of stdstr; if line is negative, one line is removed from the end. Removal occurs during the subsequent call to the initscr or newterm subroutine. When the subsequent call is made, the subroutine pointed to by init is called with two arguments: a WINDOW pointer to the one-line window that has been allocated and an integer with the number of columns in the window. The initialisation subroutine cannot use the LINES and COLS external variables and cannot call the wrefresh or doupdate subroutine, but may call the wnoutrefresh subroutine.

Up to five lines can be ripped off. Calls to the ripoffline subroutine above this limit have no effect, but report success.

Parameters

line
*init
columns
*win

Return Values

The ripoffline subroutine returns OK.

Example

To remove three lines from the top of the screen, enter:

```c
#include <curses.h>
ripoffline(1,initfunc);
ripoffline(1,initfunc);
ripoffline(1,initfunc);
initscr();
```

Related Information

The doupdate subroutine, slk_attroff, slk_init subroutine, slk_attoff, slk_att_on, slk_attron, slk_atrset, slk_attr_set, slk_clear, slk_color, slk_init, slk_label, slk_noutrefresh, slk_refresh, slk_restore, slk_set, slk_touch, slk_wset, Subroutine, initscr in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

savetty Subroutine

Purpose

Saves the state of the tty modes.

Library

Curses Library (libcurses.a)
Syntax
#include <curses.h>
savetty();

Description
The `savetty` subroutine saves the current state of the tty modes in a buffer. It saves the current state in a
buffer that the `resetty` subroutine then reads to reset the tty state.

The `savetty` subroutine is called by the `initscr` subroutine and normally should not be called directly by
the program.

Related Information
The `initscr` subroutine, `resetty, savetty` subroutine.

Curses Overview for Programming, List of Curses Subroutines, Understanding Terminals with Curses in
AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

scanw, wscanw, mvscanw, or mvwscanw Subroutine

Purpose
Calls the `wgetstr` subroutine on a window and uses the resulting line as input for a scan.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

scanw(Format, Argument1, Argument2, ...)
char *Format, *Argument1, ...

wscanw(Window, Format, Argument1, Argument2, ...)
WINDOW *Window;
char *Format, *Argument1, ...

mvscanw(Line, Column, Format, Argument1, Argument2, ...)
int Line, Column;
char *Format, *Argument1, ...

mvwscanw(Window, Line, Column, Format, Argument1, Argument2, ...)
WINDOW *Window;
int Line, Column;
char *Format, *Argument1, ...

Description
The `scanw`, `wscanw`, `mvscanw`, and `mvwscanw` subroutines call the `wgetstr` subroutine on a window
and use the resulting line as input for a scan. The `mvscanw` and `mvwscanw` subroutines move the cursor
before performing the scan function. Use the `scanw` and `mvscanw` subroutines on the stdscr and the
`wscanw` and `mvwscanw` subroutines on the user-defined window.
Parameters

**Argument**
Specifies the input to read.

**Column**
Specifies the vertical coordinate to move the logical cursor to before performing the scan.

**Format**
Specifies the conversion specifications to use to interpret the input. For more information about this parameter, see the discussion of the **Format** parameter in the `scanf` subroutine.

**Line**
Specifies the horizontal coordinate to move the logical cursor to before performing the scan.

**Window**
Specifies the window to perform the scan in. You only need to specify this parameter with the `wscanw` and `mvwscanw` subroutines.

Example

The following shows how to read input from the keyboard using the `scanw` subroutine.

```c
int id;
char deptname[25];

mvprintw(5,0,"Enter your i.d. followed by the department name:
");
refresh();
scanw("%d %s", &id, deptname);
mvprintw(7,0,"i.d.: %d, Name: %s\n", id, deptname);
refresh();
```

Related Information

The `wgetstr` subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Characters with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

**scr_dump, scr_init, scr_restore, scr_set Subroutine**

**Purpose**
File input/output functions.

**Library**
Curses Library (libcurses.a)

**Syntax**

```c
#include <curses.h>

int scr_dump
(const char *filename);

int scr_init
(const char *filename);

int scr_restore
(const char *filename);

int scr_set
(const char *filename);
```

**Description**
The `scr_dump` subroutine writes the current contents of the virtual screen to the file named by `filename` in an unspecified format.
The `scr_restore` subroutine sets the virtual screen to the contents of the file named by `filename`, which must have been written using the `scr_dump` subroutine. The next refresh operation restores the screen to the way it looked in the dump file.

The `scr_init` subroutine reads the contents of the file named by `filename` and uses them to initialize the Curses data structures to what the terminal currently has on its screen. The next refresh operation bases any updates of this information, unless either of the following conditions is true:

- The terminal has been written to since the virtual screen was dumped to `filename`.
- The terminfo capabilities rmcup and nrrmc are defined for the current terminal.

The `scr_set` subroutine is a combination of `scr_restore` and `scr_init` subroutines. It tells the program that the information in the file named by `filename` is what is currently on the screen, and also what the program wants on the screen. This can be thought of as a screen inheritance function.

**Parameters**

`filename`

**Return Values**

Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

**Examples**

For the `scr_dump` subroutine:

To write the contents of the virtual screen to `/tmp/virtual.dump` file, use:

```c
scr_dump("/tmp/virtual.dump");
```

For the `scr_restore` subroutine:

To restore the contents of the virtual screen from the `/tmp/virtual.dump` file and update the terminal screen, use:

```c
scr_restore("/tmp/virtual.dump");
doupdate();
```

**Related Information**

The `doupdate` subroutine, `endwin` subroutine, `open` subroutine, `read` subroutine, `readv`, `readvx`, or `pread` subroutine, `write`, `writev` subroutine, `scr_init` subroutine, `scr_restore` subroutine.

Curses Overview for Programming, Manipulating Window Data with Curses, Understanding Terminals with Curses and List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

**scr_init Subroutine**

**Purpose**

Initializes the curses data structures from a dump file.
Library
Curses Library (libcurses.a)

Syntax

```c
#include <curses.h>

scr_init( Filename )
char *Filename;
```

Description

The `scr_init` subroutine initializes the curses data structures from a dump file. You create dump files with the `scr_dump` subroutine. If the file’s data is valid, the next screen update is based on the contents of the file rather than clearing the screen and starting from scratch. The data is invalid if the `terminfo` database boolean capability `nrrmc` is TRUE or the contents of the terminal differ from the contents of the dump file.

Note: If `nrrmc` is TRUE, avoid calling the `putp` subroutine with the `exit_ca_mode` value before calling `scr_init` subroutine in your application.

You can call the `scr_init` subroutine after the `initscr` subroutine to update the screen with the dump file contents. Using the `keypad`, `meta`, `skl_clear`, `curs_set`, `flash`, and `beep` subroutines do not affect the contents of the screen, but cause the terminal’s modification time to change.

You can allow more than one process to share screen dumps. Both processes must be run from the same terminal. The `scr_init` subroutine first ensures that the process that created the dump is in sync with the current terminal data. If the modification time of the terminal is not the same as that specified in the dump file, the `scr_init` subroutine assumes that the screen image on the terminal has changed from that in the file, and the file’s data is invalid.

If you are allowing two processes to share a screen dump, it is important to understand that one process starts up another process. The following activities happen:

- The second process creates the dump file with the `scr_init` subroutine.
- The second process exits without causing the terminal’s time stamp to change by calling the `endwin` subroutine followed by the `scr_dump` subroutine, and then the `exit` subroutine.
- Control is passed back to the first process.
- The first process calls the `scr_init` subroutine to update the screen contents with the dump file data.

Return Values

- **ERR**  Indicates the dump file’s time stamp is old or the boolean capability `nrrmc` is TRUE.
- **OK**  Indicates that the curses data structures were successfully initialized using the contents of the dump file.

Parameters

- **Filename**  Points to a dump file.

Related Information

The `scr_dump` subroutine, `scr_restore` subroutine.

Curses Overview for Programming | List of Curses Subroutines | Manipulating Window Data with Curses in AIX 5L Version 5.3 | General Programming Concepts: Writing and Debugging Programs.
scr_restore Subroutine

Purpose
Restores the virtual screen from a dump file.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

scr_restore(FileName)
char *FileName;

Description
The scr_restore subroutine restores the virtual screen from the contents of a dump file. You create a dump file with the scr_dump subroutine. To update the terminal’s display with the restored virtual screen, call the wrefresh or doupdate subroutine after restoring from a dump file.

To communicate the screen image across processes, use the scr_restore subroutine along with the scr_dump subroutine.

Return Values
ERR Indicates the content of the dump file is incompatible with the current release of curses.
OK Indicates that the virtual screen was successfully restored from a dump file.

Parameters
FileName Identifies the name of the dump file.

Example
To restore the contents of the virtual screen from the /tmp/virtual.dump file and update the terminal screen, use:
scr_restore("/tmp/virtual.dump");
doupdate();

Related Information
The scr_dump ("scr_dump, scr_init, scr_restore, scr_set Subroutine" on page 665) subroutine, scr_init ("scr_init Subroutine" on page 666) subroutine.

scrl, scroll, wscrl Subroutine

Purpose
Scrolls a Curses window.
# Library
Curses Library (libcurses.a)

# Syntax
```c
#include <curses.h>
int scr1 (int n);
int scroll (WINDOW *win);

int wscrl (WINDOW *win, int n);
```

## Description
The `scroll` subroutine scrolls `win` one line in the direction of the first line.

The `scr1` and `wscrl` subroutines scroll the current or specified window. If `n` is positive, the window scrolls `n` lines toward the first line. Otherwise, the window scrolls `-n` lines toward the last line.

These subroutines do not change the cursor position. If scrolling is disabled for the current or specified window, these subroutines have no effect. The interaction of these subroutines with the `setscrc` subroutine is currently unspecified.

## Parameters
- `*win` Specifies the window to scroll.
- `n`

## Return Values
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

## Examples
To scroll the user-defined window `my_window` up one line, enter:
```c
WINDOW *my_window;
scroll(my_window);
```

## Related Information
The `scrollok` subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Characters with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

# scrollok Subroutine

## Purpose
Enables or disables scrolling.

## Library
Curses Library (libcurses.a)
Syntax

```c
#include <curses.h>

scrollok(Window, Flag)
WINDOW *Window;
bool Flag;
```

Description

The `scrollok` subroutine enables or disables scrolling. Scrolling occurs when a program or user:

- Moves the cursor off the window’s bottom edge.
- Enters a new-line character on the last line.
- Types the last character of the last line.

If enabled, `curses` calls a refresh as part of the scrolling action on both the window and the physical display. To get the physical scrolling effect on the terminal, it is also necessary to call the `idlok` subroutine.

If scrolling is disabled, the cursor is left on the bottom line at the location where the character was entered.

Parameters

- **Flag**: Enables scrolling when set to TRUE. Otherwise, set the `Flag` parameter to FALSE to disable scrolling.
- **Window**: Identifies the window to enable or disable scrolling in.

Examples

1. To turn scrolling on in the user-defined window `my_window`, enter:
   ```c
   WINDOW *my_window;
   scrollok(my_window, TRUE);
   ```
2. To turn scrolling off in the user-defined window `my_window`, enter:
   ```c
   WINDOW *my_window;
   scrollok(my_window, FALSE);
   ```

Related Information

The `idlok` subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Characters with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

set_curterm Subroutine

Purpose

Sets the current terminal variable to the specified terminal.

Library

Curses Library (`libcurses.a`)

Curses Syntax

```c
#include <curses.h>
#include <term.h>
```
set_curterm( Newterm)
TERMINAL *Newterm;

Description
The cur_term subroutine sets the cur_term variable to the terminal specified by the Newterm parameter. The cur_term subroutine is useful when the setupterm subroutine is called more than once. The set_curterm subroutine allows the programmer to toggle back and forth between terminals.

When information for a particular terminal is no longer required, remove it using the del_curterm subroutine.

Note: The cur_term subroutine is a low-level subroutine. You should use this subroutine only if your application must deal directly with the terminfo database to handle certain terminal capabilities. For example, use this subroutine if your application programs function keys.

Parameters
Newterm Points to a TERMINAL structure. This structure contains information about a specific terminal.

Examples
To set the cur_term variable to point to the my_term terminal, use:
TERMINAL *newterm;
set_curterm(newterm);

Related Information
The setupterm subroutine.

setscrreg or wsetscrreg Subroutine

Purpose
Creates a software scrolling region within a window.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

setscrreg(Tmargin, Bmargin)
int Tmargin, Bmargin;

wsetscrreg(Window, Tmargin, Bmargin)
WINDOW *Window;
int Tmargin, Bmargin;
Description
The **setscrreg** and **wsetscrreg** subroutines create a software scrolling region within a window. Use the **setscrreg** subroutine with the stdscr and the the **wsetscrreg** subroutine with user-defined windows.

You pass the **setscrreg** subroutines values for the top line and bottom line of the region. If the **setscrreg** subroutine and **scrollok** subroutine are enabled for the region, any attempt to move off the line specified by the **Bmargin** parameter causes all the lines in the region to scroll up one line.

**Note:** Unlike the **idlok** subroutine, the **setscrreg** subroutines have nothing to do with the use of a physical scrolling region capability that the terminal may or may not have.

Parameters

- **Bmargin** Specifies the last line number in the scrolling region.
- **Tmargin** Specifies the first line number in the scrolling region (0 is the top line of the window.)
- **Window** Specifies the window to place the scrolling region in. You specify this parameter only with the **wsetscrreg** subroutine.

Examples

1. To set a scrolling region starting at the 10th line and ending at the 30th line in the stdscr, enter:
   ```c
   setscrreg(9, 29);
   ```
   **Note:** Zero is always the first line.

2. To set a scrolling region starting at the 10th line and ending at the 30th line in the user-defined window `my_window`, enter:
   ```c
   WINDOW *my_window;
   wsetscrreg(my_window, 9, 29);
   ```

Related Information

The **idlok** ["idlok Subroutine" on page 623] subroutine, **scrollok** ["scrollok Subroutine" on page 669] subroutine, **wrefresh** ["refresh or wrefresh Subroutine" on page 658] subroutine.

**setsyx Subroutine**

**Purpose**
Sets the coordinates of the virtual screen cursor.

**Library**
Curses Library (*libcurses.a*)

**Syntax**
```c
#include <curses.h>

setsyx(int Y, int X)
```
Description
The `setsyx` subroutine sets the coordinates of the virtual screen cursor to the specified row and column coordinates. If Y and X are both -1, then the `leaveok` flag is set. (`leaveok` may be set by applications that do not use the cursor.)

The `setsyx` subroutine is intended for use in combination with the `getsyx` subroutine. These subroutines should be used by a user-defined function that manipulates curses windows but wants the position of the cursor to remain the same. Such a function would do the following:

- Call the `getsyx` subroutine to obtain the current virtual cursor coordinates.
- Continue processing the windows.
- Call the `wnoutrefresh` subroutine on each window manipulated.
- Call the `setsyx` subroutine to reset the current virtual cursor coordinates to the original values.
- Refresh the display by calling the `doupdate` subroutine.

Parameters

- `X` Specifies the column to set the virtual screen cursor to.
- `Y` Specifies the row to set the virtual screen cursor to.

Related Information

The `doupdate` subroutine, `getsyx` subroutine, `leaveok` subroutine, `wnoutrefresh` subroutine.

---

set_term Subroutine

Purpose
Switches between screens.

Library
Curses Library (`libcurses.a`)

Syntax

```c
#include <curses.h>

SCREEN *set_term
(SCREEN *new);
```

Description

The `set_term` subroutine switches between different screens. The `new` argument specifies the current screen.
Parameters

*new

Return Values
Upon successful completion, the set_term subroutine returns a pointer to the previous screen. Otherwise, it returns a null pointer.

Examples
To make the terminal stored in the user-defined SCREEN variable my_terminal the current terminal and then store a pointer to the old terminal in the user-defined variable old_terminal, enter:

SCREEN *old_terminal, *my_terminal;
old_terminal = set_term(my_terminal);

Related Information
The initscr subroutine, newterm subroutine.

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Understanding Terminals with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

setupterm Subroutine

Purpose
Initializes the terminal structure with the values in the terminfo database.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
#include <term.h>

setupterm(Term, FileNumber, ErrorCode)
char *Term;
int FileNumber;
int *ErrorCode;

Description
The setupterm subroutine determines the number of lines and columns available on the output terminal. The setupterm subroutine calls the termdef subroutine to define the number of lines and columns on the display. If the termdef subroutine cannot supply this information, the setupterm subroutine uses the values in the terminfo database.
The `setupterm` subroutine initializes the terminal structure with the terminal-dependent capabilities from `terminfo`. This routine is automatically called by the `initscr` and `newterm` subroutines. The `setupterm` subroutine deals directly with the `terminfo` database.

Two of the terminal-dependent capabilities are the lines and columns. The `setupterm` subroutine populates the lines and column fields in the terminal structure in the following manner:

1. If the environment variables `LINES` and `COLUMNS` are set, the `setupterm` subroutine uses these values.
2. If the environment variables are not set, the `setupterm` subroutine obtains the lines and columns information from the tty subsystem.
3. As a last resort, the `setupterm` subroutine uses the values defined in the `terminfo` database.

   **Note:** These may or may not be the same as the values in the `terminfo` database.

The simplest call is `setupterm((char*)0, 1, (int*)0)`, which uses all defaults.

After the call to the `setupterm` subroutine, the `cur_term` global variable is set to point to the current structure of terminal capabilities. A program can use more than one terminal at a time by calling the `setupterm` subroutine for each terminal and then saving and restoring the `cur_term` variable.

### Parameters

- **ErrorCode**
  - Specifies a pointer to an integer to return the error code to. If a null pointer (0) is passed for this parameter, no status is returned. An error causes the `setupterm` subroutine to print an error message and exit instead of returning.

- **FileNumber**
  - Specifies the output files file descriptor (1 equals standard output).

- **Term**
  - Specifies the terminal name. If 0 is passed for this parameter, the value of the `$TERM` environment variable is used.

### Return Values

One of the following status values is stored into the integer pointed to by the `ErrorCode` parameter:

- **1** Successful completion.
- **0** No such terminal.
- **-1** An error occurred while locating the `terminfo` database.

### Example

To determine the current terminal’s capabilities using `$TERM` as the terminal name, standard output as output, and returning no error codes, enter:

```
setupterm((char*)0, 1, (int*)0);
```

### Related Information

- The `termdef` ([termdef Subroutine” on page 395](#)) subroutine.

[Curses Overview for Programming](#) [List of Curses Subroutines](#) [Understanding Terminals with Curses](#) in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
_showstring Subroutine

Purpose

Dumps the string in the specified string address to the terminal at the specified location.

Library

Curses Library (libcurses.a)

Syntax

```
#include <curses.h>

_showstring(Line, Column, First, Last, String)
int Line, Column, First, Last;
char *String;
```

Description

The _showstring subroutine dumps the string in the specified string address to the terminal at the specified location. This is an internal extended curses subroutine and should not normally be called directly by the program.

Parameters

- **Column**: Specifies the horizontal coordinate of the terminal at which to dump the string.
- **First**: Specifies the beginning string address of the string to dump to the terminal.
- **Last**: Specifies the end string address of the string to dump to the terminal.
- **Line**: Specifies the vertical coordinate of the terminal at which to dump the string.
- **String**: Specifies the string to dump to the terminal.

Related Information

[Curses Overview for Programming][1], [List of Curses Subroutines][2], [Manipulating Characters with Curses][3] in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

slk_attroff, slk_attr_off, slk_attron, slk_attrset, slk_attr_set, slk_clear, slk_color, slk_init, slk_label, slk_noutrefresh, slk_refresh, slk_restore, slk_set, slk_touch, slk_wset, Subroutine

Purpose

Soft label subroutines.

Library

Curses Library (libcurses.a)

Syntax

```
#include <curses.h>

int slk_attroff
(const chtype attrs);

int slk_attr_off
```
(const attr_t attrs,
 void *opts);

int slk_attron
(const chtype attrs);

int slk_attr_on
(const attr_t attrs,
 void *opts);

int slk_attrset
(const chtype attrs);

int slk_attr_set
(const attr_t attrs,
 short color_pair_number,
 void *opts);

int slk_clear
(void);

int slk_color
(short color_pair_number);

int slk_init
(int fmt);

char *slk_label
(int labnum);

int slk_noutrefresh
(void);

int slk_refresh
(void);

int slk_restore
(void);

int slk_set
(int labnum,
 const char *label,
 int justify);

int slk_touch
(void);

int slk_wset
(int labnum,
 const wchar_t *label,
 int justify);

Description

The Curses interface manipulates the set of soft function-key labels that exist on many terminals. For those terminals that do not have sort labels, Curses takes over the bottom line of stdscr, reducing the size of stdscr and the value of the LINES external variable. There can be up to eight labels of up to eight display columns each.

To use soft labels, the slk_init subroutine must be called before initscr, newterm, or ripoffline is called. If initscr eventually uses a line from stdscr to emulate the soft labels, then fmt determines how the labels are arranged on the screen. Setting fmt to 0 indicates a 3-2-3 arrangement of the labels; 1 indicates a 4-4 arrangement. Other values for fmt are unspecified.
The **slk_init** subroutine has the effect of calling the **ripoffline** subroutine to reserve one screen line to accommodate the requested format.

The **slk_set** and **slk_wset** subroutines specify the text of soft label number `labnum`, within the range from 1 to and including 8. The `label` argument is the string to be put on the label. With **slk_set** and **slk_wset**, the width of the label is limited to eight column positions. A null string or a null pointer specifies a blank label. The `justify` argument can have the following values to indicate how to justify label within the space reserved for it:

0  Align the start of label with the start of the space.
1  Center label within the space.
2  Align the end of label with the end of the space.

The **slk_refresh** and **slk_noutrefresh** subroutines correspond to the **wrefresh** and **wnoutrefresh** subroutines.

The **slk_label** subroutine obtains soft label number `labnum`.

The **slk_clear** subroutine immediately clears the soft labels from the screen.

The **slk_touch** subroutine forces all the soft labels to be output the next time **slk_noutrefresh** or **slk_refresh** subroutines is called.

The **slk_attron**, **slk_attrset** and **slk_attroff** subroutines correspond to the **attron**, **attrset**, and **attroff** subroutines. They have an effect only if soft labels are simulated on the bottom line of the screen.

The **slk_attr_off**, **slk_attr_on**, **slk_str_set**, and **slk_attroff** subroutines correspond to the **slk_attroff**, **slk_attron**, **slk_attrset**, and **color_set** and thus support the attribute constants with the WA_prefix and color.

The `opts` argument is reserved for definition in a future edition of this document. Currently, the application must provide a null pointer as `opts`.

**Parameters**

- `attris`
- `*opts`
- `color_pair_number`
- `fmt`
- `labnum`
- `justify`
- `*label`

**Examples**

For the **slk_init** subroutine:

To initialize soft labels on a terminal that does not support soft labels internally, do the following:

```
slk_init(1);
```

This example arranges the labels so that four labels appear on the right of the screen and four appear on the left.

For the **slk_label** subroutine:
To obtain the label name for soft label 3, use:

```c
char *label_name;
label_name = slk_label(3);
```

For the `slk_noutrefresh` subroutine:

To refresh soft label 8 on the virtual screen but not on the physical screen, use:

```c
slk_set(8, "Insert", 1);
slk_noutrefresh();
```

For the `slk_refresh` subroutine:

To set and left-justify the soft labels and then refresh the physical screen, use:

```c
slk_init(0);
initscr();
slk_set(1, "Insert", 0);
slk_set(2, "Quit", 0);
slk_set(3, "Add", 0);
slk_set(4, "Delete", 0);
slk_set(5, "Undo", 0);
slk_set(6, "Search", 0);
slk_set(7, "Replace", 0);
slk_set(8, "Save", 0);
slk_refresh();
```

For the `slk_set` subroutine:

```c
slk_set(2, "Quit", 1);
```

**Return Values**

Upon successful completion, the `slk_label` subroutine returns the requested label with leading and trailing blanks stripped. Otherwise, it returns a null pointer.

Upon successful completion, the other subroutines return OK. Otherwise, they return ERR.

**Related Information**

The `attroff` subroutine, `attron` subroutine, `attrset` subroutine, `wattroff` subroutine, `wattron` subroutine, and `wattrset` subroutine.

The `ripoffline` subroutine.

The `wcswidth` subroutine.

The `slk_init` subroutine.

The `slk_set` subroutine.

**Curses Overview for Programming** | **List of Curses Subroutines** | **Manipulating Video Attributes**

In AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

**slk_init Subroutine**

**Purpose**

Initializes soft function-key labels.

**Library**

Curses Library (`libcurses.a`)

**Syntax**

```c
#include <curses.h>
```
The `slk_init` subroutine initializes soft function-key labels. This is one of several subroutines curses provides for manipulating soft function-key labels. These labels appear at the bottom of the screen and give applications, such as editors, a more user-friendly look. To use soft labels, you must call the `slk_init` subroutine before calling the `initscr` or `newterm` subroutine.

Some terminals support soft labels, others do not. For terminals that do not support soft labels, Curses emulates soft labels by using the bottom line of the stdscr. To accommodate soft labels, curses reduces the size of the stdscr and the `LINES` environment variable as required.

**Parameter**

`Labfmt` Simulates soft labels. To arrange three labels on the right, two in the center, and three on the right of the screen, specify a 0 for this parameter. To arrange four labels on the left and four on the right of the screen, specify a 1 for this parameter.

**Example**

To initialize soft labels on a terminal that does not support soft labels internally, do the following:

```
slk_init(1);
```

This example arranges the labels so that four labels appear on the right of the screen and four appear on the left.

**Related Information**

The `initscr` subroutine, `newterm` subroutine.

---

### slk_label Subroutine

**Purpose**

Returns the label name for a specified soft label.

**Library**

Curses Library (`libcurses.a`)

**Syntax**

```
#include <curses.h>

char *slk_label(int LabNum);
```

**Description**

The `slk_label` subroutine returns the label name for a specified soft function-key label. These labels appear at the bottom of the screen and give applications, such as editors, a more user-friendly look. The
The `slk_label` subroutine returns the name in the format it was in when passed to the `slk_set` subroutine. If the name was justified by the `slk_set` subroutine, the justification is removed.

**Parameters**

*LabNum* Specifies the label number. This parameter must be in the range 1 to 8.

**Example**

To obtain the label name for soft label 3, use:

```c
char *label_name;
label_name = slk_label(3);
```

**Return Values**

- **NULL** Indicates a label number that is not valid or a label number not set with the `slk_set` subroutine.
- **OK** Indicates that the label name was successfully retrieved.

**Related Information**

The `slk_init` subroutine and `slk_set` subroutine. See also *Manipulating Video Attributes* in *AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs*.

---

### slk_noutrefresh Subroutine

**Purpose**

Updates the soft labels on the virtual screen.

**Library**

Curses Library (*libcurses.a*)

**Syntax**

```c
#include <curses.h>
slk_noutrefresh();
```

**Description**

The `slk_noutrefresh` subroutine updates the soft function-key labels on the virtual screen. These labels appear at the bottom of the screen and give applications, such as editors, a more user-friendly look. This subroutine is useful for updating multiple labels. You can use the `slk_noutrefresh` subroutine to update all soft labels on the virtual screen with no updates to the physical screen. To update the physical screen, use the `slk_refresh` or `refresh` subroutine.

**Example**

To refresh soft label 8 on the virtual screen but not on the physical screen, use:

```c
slk_set(8, "Insert", 1);
slk_noutrefresh();
```
slk_refresh Subroutine

Purpose
Updates soft labels on the virtual and physical screens.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
slk_refresh()

Description
The slk_refresh subroutine refreshes the virtual and physical screens after an update to soft function-key labels. These labels appear at the bottom of the screen and give applications, such as editors, a more user-friendly look.

Example
To set and left-justify the soft labels and then refresh the physical screen, use:

slk_init(0);
inscr();
slk_set(1, "Insert", 0);
slk_set(2, "Quit", 0);
slk_set(3, "Add", 0);
slk_set(4, "Delete", 0);
slk_set(5, "Undo", 0);
slk_set(6, "Search", 0);
slk_set(7, "Replace", 0);
slk_set(8, "Save", 0);
slk_refresh();

Related Information
The slk_init routine, slk_set routine, slk_noutrefresh subroutine.

slk_restore Subroutine

Purpose
Restores soft function-key labels to the screen.
Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

slk_restore()

Description
The slk_restore subroutine restores the soft function-key labels to the screen after a call to the slk_clear subroutine. The label names are not restored. These labels appear at the bottom of the screen and give applications, such as editors, a more user-friendly look. You must call the slk_init subroutine before you can use soft labels.

Related Information
The slk_init subroutine, slk_clear subroutine.

Curses Overview for Programming List of Curses Subroutines Manipulating Soft Labels in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

slk_set Subroutine

Purpose
Sets up soft function-key labels.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

slk_set(LabNum, LabStr, LabFmt)

int LabNum;
char *LabStr;
int LabFmt;

Description
The slk_set subroutine sets up each soft function-key label with the appropriate name. These labels appear at the bottom of the screen and give applications, such as editors, a more user-friendly look. Label names are restricted to 8 characters each.

Parameters
LabNum Specifies the label number. The value can range from 1 to 8.
LabStr Specifies the string (name) to put on the label. If the string is NULL, the label is blank.
LabFmt Specifies the label alignment. The following values are valid:

<table>
<thead>
<tr>
<th>Value</th>
<th>Alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Left-justified</td>
</tr>
<tr>
<td>1</td>
<td>Centered</td>
</tr>
<tr>
<td>2</td>
<td>Right-justified</td>
</tr>
</tbody>
</table>

Example

```c
slk_set(2, "Quit", 1);
```

Related Information

The `slk_init` subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Video Attributes in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

**slk_touch Subroutine**

**Purpose**
Forces an update of the soft function-key labels.

**Library**
Curses Library (`libcurses.a`)

**Syntax**
```
#include <curses.h>

slk_touch()
```

**Description**

The `slk_touch` subroutine forces an update of the soft function-key labels on the physical screen the next time the `slk_noutrefresh` subroutine is called. These labels appear at the bottom of the screen and give applications, such as editors, a more user-friendly look. You must call the `slk_init` subroutine before using soft labels.

Related Information

The `slk_init` subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Video Attributes in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

**standend, standout, wstandend, or wstandout Subroutine**

**Purpose**
Sets and clears window attributes.

**Library**
Curses Library (`libcurses.a`)
Syntax

```c
#include <curses.h>

int standend (void);
int standout (void);
int wstandend (WINDOW *win);
int wstandout (WINDOW *win);
```

Description

The `standend` and `standout` subroutines turn off all attributes of the current or specified window.

The `wstandout` and `wstandend` subroutines turn on the `standout` attribute of the current or specified window.

Parameters

`*win` Specifies the window in which to set the attributes.

Return Values

These subroutines always return 1.

Examples

1. To turn on the `standout` attribute in the stdscr, enter:
   ```c
   standout();
   ```
   This example is functionally equivalent to:
   ```c
   attron(A_STANDOUT);
   ```

2. To turn on the `standout` attribute in the user-defined window `my_window`, enter:
   ```c
   WINDOW *my_window;
   wstandout(my_window);
   ```
   This example is functionally equivalent to:
   ```c
   wattron(my_window, A_STANDOUT);
   ```

3. To turn off the `standout` attribute in the default window, enter:
   ```c
   standend();
   ```
   This example is functionally equivalent to:
   ```c
   attroff(A_STANDOUT);
   ```

4. To turn off the `standout` attribute in the user-defined window `my_window`, enter:
   ```c
   WINDOW *my_window;
   wstandend(my_window);
   ```
   This example is functionally equivalent to:
Related Information

The `attroff`, `attron`, or `wattroff` subroutines.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Video Attributes in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

**start_color Subroutine**

**Purpose**

Initializes color.

**Library**

Curses Library (`libcurses.a`)

**Syntax**

```
#include <curses.h>
start_color()
```

**Description**

The `start_color` subroutine initializes color. This subroutine requires no arguments. You must call the `start_color` subroutine if you intend to use color in your application. Except for the `has_colors` and `can_change_color` subroutines, you must call the `start_color` subroutine before any other color manipulation subroutine. A good time to call `start_color` is right after calling the `initscr` routine and after establishing whether the terminal supports color.

The `start_color` routine initializes the following basic colors:

<table>
<thead>
<tr>
<th>Color</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR_BLACK</td>
<td>0</td>
</tr>
<tr>
<td>COLOR_BLUE</td>
<td>1</td>
</tr>
<tr>
<td>COLOR_GREEN</td>
<td>2</td>
</tr>
<tr>
<td>COLOR_CYAN</td>
<td>3</td>
</tr>
<tr>
<td>COLOR_RED</td>
<td>4</td>
</tr>
<tr>
<td>COLOR_MAGENTA</td>
<td>5</td>
</tr>
<tr>
<td>COLOR_YELLOW</td>
<td>6</td>
</tr>
<tr>
<td>COLOR_WHITE</td>
<td>7</td>
</tr>
</tbody>
</table>

The subroutine also initializes two global variables: COLORS and COLOR_PAIRS. The COLORS variable is the maximum number of colors supported by the terminal. The COLOR_PAIRS variable is the maximum number of color-pairs supported by the terminal.

The `start_color` subroutine also restores the terminal’s colors to the original values right after the terminal was turned on.

**Return Values**

- **ERR** Indicates the terminal does not support colors.
- **OK** Indicates the terminal does support colors.
Example
To enable the color support for a terminal that supports color, use:

\texttt{start\_color();}

Related Information
The \texttt{has\_colors} subroutine, \texttt{can\_change\_color} subroutine, \texttt{can\_change\_color, color\_content, has\_colors, init\_color, init\_pair, start\_color or pair\_content Subroutine} subroutine.

Curses Overview for Programming \textbf{List of Curses Subroutines} Manipulating Video Attributes in \textit{AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.}

---

subpad Subroutine

Purpose
Creates a subwindow within a pad.

Library
Curses Library (libcurses.a)

Syntax

\begin{verbatim}
#include <curses.h>

WINDOW *subpad(WINDOW *Orig, int NLines, int NCols, int Begin_Y, int Begin_X)
\end{verbatim}

Description
The \texttt{subpad} subroutine creates and returns a pointer to a subpad. A subpad is a window within a pad. You specify the size of the subpad by supplying a starting coordinate and the number of rows and columns within the subpad. Unlike the \texttt{subwin} subroutine, the starting coordinates are relative to the pad and not the terminal's display.

Changes to the subpad affect the character image of the parent pad, as well. If you change a subpad, use the \texttt{touchwin} or \texttt{touchline} subroutine on the parent pad before refreshing the parent pad. Use the \texttt{prefresh} subroutine to refresh a pad.

Parameters

- \texttt{Orig} Points to the parent pad.
- \texttt{NLines} Specifies the number of lines (rows) in the subpad.
- \texttt{NCols} Specifies the number of columns in the subpad.
- \texttt{Begin\_Y} Identifies the upper left-hand row coordinate of the subpad relative to the parent pad.
- \texttt{Begin\_X} Identifies the upper left-hand column coordinate of the subpad relative to the parent pad.

Examples
To create a subpad, use:

\begin{verbatim}
WINDOW *orig, *mypad;
orig = newpad(100, 200);
mypad = subpad(orig, 30, 5, 25, 180);
\end{verbatim}
The parent pad is 100 lines by 200 columns. The subpad is 30 lines by 5 columns and starts in line 25, column 180 of the parent pad.

**Related Information**

- Curses Overview for Programming
- List of Curses Subroutines
- Windows in the Curses Environment in AIX 5L Version 5.3
- General Programming Concepts: Writing and Debugging Programs.

---

**subwin Subroutine**

**Purpose**

Creates a subwindow within an existing window.

**Library**

Curses Library (libcurses.a)

**Syntax**

```c
#include <curses.h>
WINDOW *subwin (ParentWindow, NumLines, NumCols, Line, Column)
```

**Description**

The `subwin` subroutine creates a subwindow within an existing window. You must supply coordinates for the subwindow relative to the terminal’s display. Recall that the subwindow shares its parent’s window buffer. Changes made to the shared window buffer in the area covered by a subwindow, through either the parent window or any of its subwindows, affects all windows sharing the window buffer.

When changing the image of a subwindow, it is necessary to call the `touchwin` subroutine on the parent window before calling the `wrefresh` subroutine on the parent window.

Changes to one window will affect the character image of both windows.

**Parameters**

- **NumCols**
  Indicates the number of vertical columns in the subwindow’s width. If 0 is passed as the `NumCols` value, the subwindow runs from the Column to the right edge of its parent window.

- **NumLines**
  Indicates the number of horizontal lines in the subwindow’s height. If 0 is passed as the `NumLines` parameter, then the subwindow runs from the Line to the bottom of its parent window.

- **ParentWindow**
  Specifies the subwindow’s parent.

- **Column**
  Specifies the horizontal coordinate for the upper-left corner of the subwindow. This coordinate is relative to the (0, 0) coordinates of the terminal, not the (0, 0) coordinates of the parent window.
  
  **Note:** The upper-left corner of the terminal is referenced by the coordinates (0, 0).

- **Line**
  Specifies the vertical coordinate for the upper-left corner of the subwindow. This coordinate is relative to the (0, 0) coordinates of the terminal, not the (0, 0) coordinates of the parent window.
  
  **Note:** The upper-left corner of the terminal is referenced by the coordinates (0, 0).
Return Values
When the subwin subroutine is successful, it returns a pointer to the subwindow structure. Otherwise, it returns the following:

ERR Indicates one or more of the parameters is invalid or there is insufficient storage available for the new structure.

Examples
1. To create a subwindow, use:

```
WINDOW *my_window, *my_sub_window;

my_window = newwin ("derwin, newwin, or subwin Subroutine" on page 646)
(5, 10, 20, 30);

my_sub_window = subwin(my_window, 2, 5, 20, 30);
```

my_sub_window is now a subwindow 2 lines deep, 5 columns wide, starting at the same coordinates of its parent window my_window. That is, the subwindow's upper-left corner is at coordinates y = 20, x = 30 and lower-right corner is at coordinates y = 21, x = 34.

2. To create a subwindow that is flush with the right side of its parent, use:

```
WINDOW *my_window, *my_sub_window;

my_window = newwin ("derwin, newwin, or subwin Subroutine" on page 646)
(5, 10, 20, 30);

my_sub_window = subwin(my_window, 2, 0, 20, 30);
```

my_sub_window is now a subwindow 2 lines deep, extending all the way to the right side of its parent window my_window, and starting at the same coordinates. That is, the subwindow's upper-left corner is at coordinates y = 20, x = 30 and lower-right corner is at coordinates y = 21, x = 39.

3. To create a subwindow in the lower-right corner of its parent, use:

```
WINDOW *my_window, *my_sub_window

my_window = newwin ("derwin, newwin, or subwin Subroutine" on page 646)
(5, 10, 20, 30);

my_sub_window = subwin(my_window, 0, 0, 22, 35);
```

my_sub_window is now a subwindow that fills the bottom right corner of its parent window, my_window, starting at the coordinates y = 22, x = 35. That is, the subwindow's upper-left corner is at coordinates y = 22, x = 35 and lower-right corner is at coordinates y = 24, x = 39.

Related Information
The touchwin ("touchwin Subroutine" on page 700), newwin ("derwin, newwin, or subwin Subroutine" on page 646), and wrefresh ("refresh or wrefresh Subroutine" on page 658) subroutines.

Curses Overview for Programming List of Curses Subroutines Windows in the Curses Environment in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
tgetent, tgetflag, tgetnum, tgetstr, or tgoto Subroutine

Purpose
Termcap database emulation.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

int tgetent
(char *bp,
 const char *name);

int tgetflag
(char id[2]);

int tgetnum
(char id[2]);

char *tgetstr
(char id[2],
 char **area);

char *tgoto
(char *cap,
 int col,
 int row);

Description
The tgetent subroutine looks up the termcap entry for name, The emulation ignores the buffer pointer bp.

The tgetflag subroutine gets the boolean entry for id.

The tgetnum subroutine gets the numeric entry for id.

The tgetstr subroutine gets the string entry for id. If area is not a null pointer and does not point to a null pointer, the tgetstr subroutine copies the string entry into the buffer pointed to by *area and advances the variable pointed to by area to the first byte after the copy of the string entry.

The tgoto subroutine instantiates the parameters col and row into the capability cap and returns a pointer to the resulting string.

All of the information available in the terminfo database need not be available through these subroutines.

Parameters
bp
name
col
row
**area
cap id[2]
**Return Values**
Upon successful completion, subroutines that return an integer return OK. Otherwise, they return ERR.

**Related Information**
The `putc` subroutine ("setupterm Subroutine" on page 674), `tigetflag` ("tigetflag, tigetnum, tigetstr, or tparm Subroutine" on page 694) subroutines.

**tgetflag Subroutine**

**Purpose**
Returns the boolean entry for the specified identifier.

**Library**
Curses Library (`libcurses.a`)

**Syntax**
```
#include <curses.h>

bool tgetflag(ID)
char *ID;
```

**Description**
The `tgetflag` subroutine returns the boolean entry for the specified `termcap` identifier. This subroutine is provided for binary compatibility with applications that use the `termcap` file.

**Parameters**

*ID*  Specifies the 2-character string that contains a `termcap` identifier.

**Return Values**
The `tgetflag` subroutine returns the boolean entry for the specified `termcap` identifier. If *ID* is not found, on not a boolean, 0 is returned.

**Related Information**
[Curses Overview for Programming](AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs)
[List of Curses Subroutines](AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs)
[Understanding Terminals with Curses](AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs)
tgetnum Subroutine

Purpose
Returns the numeric entry for the specified termcap identifier.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>

int tgetnum(
char *ID;

Description
The tgetnum subroutine returns the numeric entry for the specified termcap identifier. This subroutine is provided for binary compatibility with applications that use the termcap file.

Parameters
ID Specifies the 2-character string that contains a termcap identifier.

Return Values
The tgetnum subroutine returns the numeric entry for the specified termcap identifier.
-1 Returned if the ID is not found or not numeric.

Related Information
Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
Understanding Terminals with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

tgetstr Subroutine

Purpose
Returns the string entry for the specified termcap identifier.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
char *tgetstr(ID, Area);
char *ID, **Area;

Description
The tgetstr subroutine returns the string entry for the specified termcap identifier. This subroutine is provided for binary compatibility with applications that use the termcap file.

Parameters
Area Contains the string entry for the specified termcap identifier. This also is returned to the calling program.
ID Specifies the 2-character string that contains the termcap identifier.

Return Values
The tgetstr subroutine returns the string entry for the ID parameter, which is a 2-character string that contains a termcap identifier.

0 Returned if ID is not found or not a string capability.

Related Information
Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
Understanding Terminals with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

tgoto Subroutine

Purpose
Duplicates the tparm subroutine.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
#include <term.h>

char *tgoto(Capability, Column, Row);
char *Capability;
int Column, Row;

Description
The tgoto subroutine calls the tparm subroutine. This subroutine is provided for binary compatibility with applications that use the termcap file.
Parameters

- **Capability**: Specifies the termcap capability to apply the parameters to.
- **Column**: Specifies which column to apply to the capability.
- **Row**: Specifies which row to apply to the capability.

Related Information

The `tparm` subroutine.

Curses Overview for Programming, List of Curses Subroutines, Understanding Terminals with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

tigetflag, tigetnum, tigetstr, or tparm Subroutine

Purpose

Retrieves capabilities from the `terminfo` database.

Library

Curses Library (`libcurses.a`)

Syntax

```c
#include <term.h>

int tigetflag(char *capname);
int tigetnum(char *capname);
char *tigetstr(char *capname);
char *tparm(char *cap,
long p1, long p2, long p3,
long p4, long p5, long p6
long p7, long p8, long p9);
```

Description

The `tigetflag`, `tigetnum`, and `tigetstr` subroutines obtain boolean, numeric, and string capabilities, respectively, from the selected record of the terminfo database. For each capability, the value to use as `capname` appears in the Capname column in the table in Section 6.1.3 on page 296.

The `tparm` subroutine takes as `cap` a string capability. If `cap` is parameterised (as described in Section A.1.2 on page 313), the `tparm` subroutine resolves the parameterisation. If the parameterised string refers to parameters `%p1 through %p9`, then the `tparm` subroutine substitutes the values of `p1` through `p9`, respectively.

Return Values

Upon successful completion, the `tigetflag`, `tigetnum`, and `tigetstr` subroutines return the specified capability. The `tigetflag` subroutine returns -1 if `capname` is not a boolean capability. The `tigetnum` subroutine returns -2 if `capname` is not a numeric capability. The `tigetstr` subroutine returns (char*)-1 if `capname` is not a string capability.

Upon successful completion, the `tparm` subroutine returns `str` with parameterisation resolved. Otherwise, it returns a null pointer.
Parameters

*capname
*tparm
long p1
long p2
long p3
long p4
long p5
long p6
long p7
long p8
long p9

Examples
For the tigetflag subroutine:

To determine if erase overstrike is a defined boolean capability for the current terminal, use:
rc = tigetflag("eo");

For the tigetnum subroutine:

To determine if number of labels is a defined numeric capability for the current terminal, use:
rc = tigetnum("nlab");

For the tigetstr subroutine:

To determine if "turn on soft labels" is a defined string capability for the current terminal, do the following:
char *rc;
rc = tigetstr("smln");

For the tparm subroutine:
1. To save the escape sequence used to home the cursor in the user-defined variable home_sequence, enter:
home_sequence = tparm(cursor_home);
2. To save the escape sequence used to move the cursor to the coordinates X=40, Y=18 in the user-defined variable move_sequence, enter:
move_sequence = tparm(cursor_address, 18, 40);

Related Information
The def_prog_mode ("def_prog_mode, def_shell_mode, reset_prog_mode or reset_shell_mode Subroutine" on page 595), tgetent ("tgetent, tgetflag, tgetnum, tgetstr, or tgoto Subroutine" on page 690), and putp ("putp, tputs Subroutine" on page 655) subroutines.

Curses Overview for Programming List of Curses Subroutines
Understanding Terminals with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
## tigetnum Subroutine

### Purpose
Gets the value of terminal’s numeric capability.

### Library
Curses Library (libcurses.a)

### Syntax
```c
#include <curses.h>
#include <term.h>

tigetnum(CapName);
register char *CapName;
```

### Description
The `tigetnum` subroutine returns the value of terminal’s numeric capability. Use this subroutine to get a capability for the current terminal. When successful, this subroutine returns the current value of the capability specified by the `CapName` parameter. Otherwise, if it is not a numeric value, this subroutine returns -2.

**Note:** The `tigetnum` subroutine is a low-level routine. Use this subroutine only if your application must deal directly with the terminfo database to handle certain terminal capabilities (for example, programming function keys).

### Return Values
Upon successful completion, the `tigetnum` subroutine returns the value of terminal’s numeric capability.

-2 Indicates the value specified by the `CapName` parameter is not numeric.

### Parameters
`CapName` Identifies the terminal capability to check for.

### Example
To determine if number of labels is a defined numeric capability for the current terminal, use:
```c
rc = tigetnum("nlab");
```

### Related Information
- [Curses Overview for Programming](#) in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
- [List of Curses Subroutines](#) in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
- [Understanding Terminals with Curses](#) in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
tigetstr Routine

Purpose
Returns the value of a terminal's string capability.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
#include <term.h>

tigetstr( Capname )
register char *Capname;

Description
The tigetstr subroutine returns the value of terminal's string capability. Use this subroutine to get a capability for the current terminal pointed to by cur_term. When successful, this subroutine returns the current value of the capability specified by the Capname parameter. Otherwise, if it is not a string value, this subroutine returns (char*) -1.

Note: The tigetstr subroutine is a low-level routine. Use this subroutine only if your application must deal directly with the terminfo database to handle certain terminal capabilities (for example, programming function keys).

Parameters
Capname Identifies the terminal capability to check.

Example
To determine if "turn on soft labels" is a defined string capability for the current terminal, do the following:
  char *rc;
  rc = tigetstr("smln");

Return Values
Upon successful completion, the tigetstr subroutine returns the value of terminal's string capability.

(char*)-1 Indicates the value specified by the Capname parameter is not a string.

Files
/usr/include/curses.h Contains C language subroutines and define statements for curses.

Related Information
List of Curses Subroutines Curses Overview for Programming Understanding Terminals with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
**is_linetouched, is_wintouched, touchline, touchwin, untouchwin, or wtouchin Subroutine**

**Purpose**
Window refresh control functions.

**Library**
Curses Library (*libcurses.a*)

**Syntax**
```c
#include <curses.h>

bool is_linetouched(WINDOW *win, int line);
bool is_wintouched(WINDOW *win);

int touchline(WINDOW *win, int start, int count);
int touchwin(WINDOW *win);
int untouchwin(WINDOW *win);
int wtouchln(WINDOW *win, int y, int n, int changed);
```

**Description**
The `touchline` subroutine touches the specified window (that is, marks it as having changed more recently than the last refresh operation). The `touchline` subroutine only touches count lines, beginning with line `start`.

The `untouchwin` subroutine marks all lines in the window as unchanged since the last refresh operation.

Calling the `wtouchln` subroutine, if changed is 1, touches n lines in the specified window, starting at line `y`. If changed is 0, `wtouchln` marks such lines as unchanged since the last refresh operation.

The `is_wintouched` subroutine determines whether the specified window is touched. The `is_linetouched` subroutine determines whether line `line` of the specified window is touched.

**Parameters**

- `line`
- `start`
- `count`
- `changed`
- `y`
- `n`
- `*win`
Return Values
The is_linetouched and is_wintouched subroutines return TRUE if any of the specified lines, or the specified window, respectively, has been touched since the last refresh operation. Otherwise, they return FALSE.

Upon successful completion, the other subroutines return OK. Otherwise, they return ERR. Exceptions to this are noted in the preceding subroutine.

Examples
For the touchline subroutine:

To set 10 lines for refresh starting from line 5 of the user-defined window my_window, use:

```c
WINDOW *my_window;
touchline(my_window, 5, 10);
wrefresh(my_window);
```

This forces curses to disregard any optimization information it may have for lines 0-4 in my_window. curses assumes all characters in lines 0-4 have changed.

For the touchwin subroutine:

To refresh a user-defined parent window, parent_window, that has been edited through its subwindows, use:

```c
WINDOW *parent_window;
touchwin(parent_window);
wrefresh(parent_window);
```

This forces curses to disregard any optimization information it may have for my_window. curses assumes all lines and columns have changed for my_window.

Related Information
The doupdate ("doupdate, refresh, wnoutrefresh, or wrefresh Subroutines" on page 707) subroutine.

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Understanding Windows with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

touchoverlap Subroutine

Purpose
Marks the overlap of two windows as changed and makes arrangements for their refresh.

Library
Curses Library (libcurses.a)
Syntax

```c
#include <curses.h>

touchoverlap(Window1, Window2)
WINDOW *Window1, Window2;
```

Description
The `touchoverlap` subroutine marks the overlap of two windows as changed and makes arrangements for their refresh.

Parameters

- `Window1`: Specifies the first window as changed.
- `Window2`: Specifies the second window as changed.

Examples
To mark the overlap of the two user-defined windows `my_window` and `my_new_window` as changed, enter:
```
touchoverlap(my_window, my_new_window);
```

Related Information
- [Curses Overview for Programming](#) in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
- [List of Curses Subroutines](#) in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
- [Understanding Windows with Curses](#) in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

---

**touchwin Subroutine**

Purpose
Forces every character in a window’s buffer to be refreshed at the next call to the `wrefresh` subroutine.

Library
Curses Library (`libcurses.a`)

Syntax

```c
#include <curses.h>

touchwin(Window)
WINDOW *Window;
```

Description
The `touchwin` subroutine forces every character in the specified window to be refreshed during the next call to the `refresh` or `wrefresh` subroutine. To force a specific range of lines to be refreshed, use the `touchline` subroutine.
The combined usage of the **touchwin** and **wrefresh** subroutines is helpful when dealing with subwindows or overlapping windows. When dealing with overlapping windows, it may become necessary to bring the back window to the front. A call to the **wrefresh** subroutine does not change the terminal because none of the characters in the window were changed. Calling the **touchwin** subroutine on the back window before the **wrefresh** subroutine redispalyes the window on the terminal and, effectively, brings it to the front.

**Parameters**

*Window* Specifies the window to be touched.

**Example**

To refresh a user-defined parent window, `parent_window`, that has been edited through its subwindows, use:

```c
WINDOW *parent_window;
touchwin(parent_window);
wrefresh(parent_window);
```

This forces **curses** to disregard any optimization information it may have for `my_window`. **curses** assumes all lines and columns have changed for `my_window`.

**Related Information**

The **touchline** ("is_linetouched, is_wintouched, touchline, touchwin, untouchwin, or wtouchin Subroutine" on page 698) subroutine, **wrefresh** ("refresh or wrefresh Subroutine" on page 658) subroutine.

**tparm Subroutine**

**Purpose**

Applies parameters (padding) to a terminal capability.

**Library**

Curses Library (**libcurses.a**)

**Syntax**

```c
#include <curses.h>

char *tparm(TermCap, Parm1, Parm2, . . . Parm9)
char *TermCap;
int Parm1, Parm2, . . . Parm9;
```

**Description**

The **tparm** subroutine applies parameters (padding) to a terminal capability.

**Note:** If the **tparm** subroutine is called with less than 10 parameters, then the **-D_TPARM_COMPAT** option should be used when compiling the program. Otherwise the compiler gives the following error.

1506-098 (E) Missing argument(s)
Parameters

 Parm# Specifies the parameters (up to nine) to instantiate.
 TermCap Specifies the terminal capability to apply the parameters to. These terminal capabilities are defined in the term.h file.

Return Values

The tparm subroutine returns the escape sequence specified by the TermCap parameter with the specified parameters applied. After the escape sequence is received, it can be output by a subroutine like the tputs subroutine.

Examples

1. To save the escape sequence used to home the cursor in the user-defined variable home_sequence, enter:
   home_sequence = tparm(cursor_home);
2. To save the escape sequence used to move the cursor to the coordinates X=40, Y=18 in the user-defined variable move_sequence, enter:
   move_sequence = tparm(cursor_address, 18, 40);

Related Information

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Understanding Terminals with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

tputs Subroutine

Purpose

Outputs a string with padding information.

Library

Curses Library (libcurses.a)

Syntax

#include <curses.h>
#include <term.h>

tputs( String, LinesAffected, PutcLikeSub)
char *String;
int LinesAffected;
int (*PutcLikeSub)();

Description

The tputs subroutine outputs a string with padding information applied. String must be a terminfo string variable or the return value from tparm, tgetstr, tigetstr, or tgoto subroutines.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinesAffected</td>
<td>Specifies the number of lines affected, or specifies 1 if not applicable.</td>
</tr>
<tr>
<td>PutcLikeSub</td>
<td>Specifies a <code>putchar</code>-like subroutine through which the characters are passed one at a time.</td>
</tr>
<tr>
<td>String</td>
<td>Specifies the string to which to add padding information.</td>
</tr>
</tbody>
</table>

Examples

1. To output the clear screen sequence using the user-defined `putchar`-like subroutine `my_putchar`, enter:
   
   ```c
   int my_putchar();
   tputs(clear_screen, 1 ,my_putchar);
   ```

2. To output the escape sequence used to move the cursor to the coordinates x=40, y=18 through the user-defined `putchar`-like subroutine `my_putchar`, enter:
   
   ```c
   int my_putchar();
   tputs(tparm(cursor_address, 18, 40), 1, my_putchar);
   ```

Related Information

The `tparm` subroutine.

Curses Overview for Programming List of Curses Subroutines Understanding Terminals with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

typeahead Subroutine

Purpose

Controls checking for typeahead.

Library

Curses Library (libcurses.a)

Syntax

```c
#include <curses.h>

int typeahead(int fildes);
```

Description

The `typeahead` subroutine controls the detection of typeahead during a refresh, based on the value of `fildes`:

- If `fildes` is a valid file descriptor, the `typeahead` subroutine is enabled during refresh; Curses periodically checks `fildes` for input and aborts refresh if any character is available. (This is the initial setting, and the `typeahead` file descriptor corresponds to the input file associated with the screen created by the `initscr` or `newterm` subroutine.) The value of `fildes` need not be the file descriptor on which the refresh is occurring.
- If `fildes` is -1, Curses does not check for typeahead during refresh.

Parameters

`fildes`
Return Value
Upon successful completion, the `typeahead` subroutine returns OK. Otherwise, it returns ERR.

Example
To turn typeahead checking on, enter:
```
typeahead(1);
```

Related Information
The `doupdate` ("doupdate, refresh, wnoutrefresh, or wrefresh Subroutines" on page 707), `getch` ("getch, mvgetch, mvwgetch, or wgetch Subroutine" on page 611), and `initscr` ("initscr and newterm Subroutine" on page 627) subroutines.

References
- Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
- List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
- Setting Video Attributes and Curses Options in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

unctrl Subroutine

Purpose
Generates a printable representation of a character.

Library
Curses Library (`libcurses.a`)

Syntax
```
#include <curses.h>

char *unctrl
(char c);
```

Description
The `unctrl` subroutine generates a character string that is a printable representation of `c`. If `c` is a control character, it is converted to the ^X notation. If `c` contains rendition information, the effect is undefined.

Parameters

`c`

Return Values
Upon successful completion, the `unctrl` subroutine returns the generated string. Otherwise, it returns a null pointer.

Examples
To display a printable representation of the newline character, enter:
char *new_line;
int my_character;
addstr ("Hit the enter key.");
my_character = getch();
new_line = unctrl (my_character);
printw (Newline=%s", new_line);
refresh();

This prints, "newline=^J".

Related Information
The keyname ("keyname, key_name Subroutine" on page 631) subroutine.

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Manipulating Characters with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

ungetch, unget_wch Subroutine

Purpose
Pushes a character onto the input queue.

Library
Curses Library (libcurses.a)

Syntax
#include <curses.h>
int ungetch
(int ch);
int unget_wch
(const wchar_t wch);

Description
The ungetch subroutine pushes the single-byte character ch onto the head of the input queue.

The unget_wch subroutine pushes the wide character wch onto the head of the input queue.

One character of push-back is guaranteed. The result of successive calls without an intervening call to the getch or get_wch subroutine are unspecified.

Parameters
ch
wch
Examples
To force the key KEY_ENTER back into the queue, use:
ungetch(KEY_ENTER);

Related Information
The `getch` or `wgetch` subroutine.

Curses Overview for Programming and List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Manipulating Characters with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

vidattr, vid_attr, vidputs, or vid_puts Subroutine

Purpose
Outputs attributes to the terminal.

Library
Curses Library (libcurses.a)

Syntax

```c
#include <curses.h>

int vidattr
(ctype attr);

int vid_attr
(attr_t attr,
short color_pair_number,
void *opt);

int vidputs
(ctype attr,
int (*putfunc)(int));

int vid_puts
(attr_t attr,
short color_pair_number,
void *opt,
int (*putfunc)(int));
```

Description
These subroutines output commands to a terminal that changes the terminal's attributes.

If the terminfo database indicates that the terminal in use can display characters in the rendition specified by `attr`, then the `vidattr` subroutine outputs one or more commands to request that the terminal display subsequent characters in that rendition. The subroutine outputs by calling the `putchar` subroutine. The `vidattr` subroutine neither relies on nor updates the model that Curses maintains of the prior rendition mode.

The `vidputs` subroutine computes the same terminal output string that `vidattr` does, based on `attr`, but the `vidputs` subroutine outputs by calling the user-supplied subroutine `putfunc`. The `vid_attr` and `vid_puts` subroutines correspond to `vidattr` and `vidputs` respectively, but take a set of arguments, one of type `attr_t` for the attributes, `short` for the color pair number and a `void *`, and thus support the attribute constants with the WA_prefix.
The opts argument is reserved for definition in a future edition of this document. Currently, the application must provide a null pointer as opts.

The user-supplied putfunc subroutine (which can be specified as an argument to either vidputs or vid_puts) is either putchar or some other subroutine with the same prototype. Both the vidputs and the vid_puts subroutines ignore the return value of putfunc.

Parameters

att
color_pair_number
*opt
*putfunc

Return Values
Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

Examples
1. To output the string that puts the terminal in its best standout mode through the putchar subroutine, enter
   vidattr(A_STANDOUT);
2. To output the string that puts the terminal in its best standout mode through the putchar-like subroutine my_putc, enter
   int (*my_putc)();
   vidputs(A_STANDOUT, my_putc);

Related Information
The doupdate ("doupdate, refresh, wnoutrefresh, or wrefresh Subroutines"), is linetouched ("is linetouched, is wintouched, touchline, touchwin, untouchwin, or wtouchin Subroutine" on page 698), putchar, putwchar and tigetflag ("tigetflag, tigetnum, tigetstr, or tparm Subroutine" on page 694) subroutines.

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Setting Video Attributes and Curses Options in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

doupdate, refresh, wnoutrefresh, or wrefresh Subroutines

Purpose

Refreshes windows and lines.

Library

Curses Library (libcurses.a)
Syntax

#include <curses.h>

int doupdate(void);
int refresh(void);
int wnoutrefresh(WINDOW *win);
int wrefresh(WINDOW *win);

Description

The refresh and wrefresh subroutines refresh the current or specified window. The subroutines position the terminal’s cursor at the cursor position of the window, except that, if the leaveok mode has been enabled, they may leave the cursor at an arbitrary position.

The wnoutrefresh subroutine determines which parts of the terminal may need updating.

The doupdate subroutine sends to the terminal the commands to perform any required changes.

Parameters

*win Specifies the window to be refreshed.

Return Values

Upon successful completion, these subroutines return OK. Otherwise, they return ERR.

Examples

For the doupdate or wnoutrefresh subroutine:

To update the user-defined windows my_window1 and my_window2, enter:
WINDOW *my_window1, my_window2;
wnoutrefresh(my_window1);
wnoutrefresh(my_window2);
doupdate();

For the refresh or wrefresh subroutine:

1. To update the terminal’s display and the current screen structure to reflect changes made to the standard screen structure, use:
   refresh();

2. To update the terminal and the current screen structure to reflect changes made to a user-defined window called my_window, use:
   WINDOW *my_window;
   wrefresh(my_window);

3. To restore the terminal to its state at the last refresh, use:
   wrefresh(curscr);

This subroutine is useful if the terminal becomes garbled for any reason.
Related Information

The clearok subroutine.

Curses Overview for Programming in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

List of Curses Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.

Manipulating Window Data with Curses in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
Chapter 3. FORTRAN Basic Linear Algebra Subroutines (BLAS)

SDOT or DDOT Function

Purpose

Returns the dot product of two vectors.

Library

BLAS Library (libblas.a)

FORTRAN Syntax

REAL FUNCTION SDOT(N, X, INCX, Y, INCY)
INTEGER INCX, INCY
REAL X(*), Y(*)

DOUBLE PRECISION FUNCTION DDOT(N, X, INCX, Y, INCY)
INTEGER INCX, INCY, N
DOUBLE PRECISION X(*), Y(*)

Description

The SDOT or DDOT function returns the dot product of vectors X and Y.

Parameters

N  On entry, N specifies the number of elements in X and Y; unchanged on exit.
X  Vector of dimension at least (1 + (N-1) * abs(INCX)); unchanged on exit.
INCY On entry, INCY specifies the increment for the elements of X; unchanged on exit.
Y  Vector of dimension at least (1 + (N-1) * abs(INCY)); unchanged on exit.
INCY On entry, INCY specifies the increment for the elements of Y; unchanged on exit.

Error Codes

For values of N <= 0, a value of 0 is returned.

CDOTC or ZDOTC Function

Purpose

Returns the complex dot product of two vectors, conjugating the first.

Library

BLAS Library (libblas.a)

FORTRAN Syntax

COMPLEX FUNCTION CDOTC(N, X, INCX, Y, INCY)
INTEGER INCX, INCY
COMPLEX X(*), Y(*)

© Copyright IBM Corp. 1994, 2005
DOUBLE COMPLEX FUNCTION ZDOTC(N, X, INCX, Y, INCY)
INTEGER INCX, INCY, N
COMPLEX*16 X(*), Y(*)

Description
The CDOTC or ZDOTC function returns the complex dot product of two vectors, conjugating the first.

Parameters
N    On entry, N specifies the number of elements in X and Y; unchanged on exit.
X    Vector of dimension at least (1 + (N-1) * abs(INCX )); unchanged on exit.
INCX On entry, INCX specifies the increment for the elements of X; unchanged on exit.
Y    Vector of dimension at least (1 + (N-1) * abs(INCY )); unchanged on exit.
INCY On entry, INCY specifies the increment for the elements of Y; unchanged on exit.

Error Codes
For values of N <= 0, a value of 0 is returned.

CDOTU or ZDOTU Function

Purpose
Returns the complex dot product of two vectors.

Library
BLAS Library (libblas.a)

FORTRAN Syntax
COMPLEX FUNCTION CDOTU(N, X, INCX, Y, INCY)
INTEGER INCX, INCY, N
COMPLEX X(*), Y(*)
DOUBLE COMPLEX FUNCTION ZDOTU(N, X, INCX, Y, INCY)
INTEGER INCX, INCY, N
COMPLEX*16 X(*), Y(*)

Description
The CDOTU or ZDOTU function returns the complex dot product of two vectors.

Parameters
N    On entry, N specifies the number of elements in X and Y; unchanged on exit.
X    Vector of dimension at least (1 + (N-1) * abs(INCX )); unchanged on exit.
INCX On entry, INCX specifies the increment for the elements of X; unchanged on exit.
Y    Vector of dimension at least (1 + (N-1) * abs(INCY )); unchanged on exit.
INCY On entry, INCY specifies the increment for the elements of Y; unchanged on exit.

Error Codes
For values of N <= 0, a value of 0 is returned.
SAXPY, DAXPY, CAXPY, or ZAXPY Subroutine

Purpose
Computes a constant times a vector plus a vector.

Library
BLAS Library (libblas.a)

FORTRAN Syntax

SUBROUTINE SAXPY(N, A, X, INCX, Y, INCY)
INTEGER INCX, INCY, N
REAL A
REAL X(*), Y(*)

SUBROUTINE DAXPY(N, A, X, INCX, Y, INCY)
INTEGER INCX, INCY, N
DOUBLE PRECISION A
DOUBLE PRECISION X(*), Y(*)

SUBROUTINE CAXPY(N, A, X, INCX, Y, INCY)
INTEGER INCX, INCY, N
COMPLEX A
COMPLEX X(*), Y(*)

SUBROUTINE ZAXPY(N, A, X, INCX, Y, INCY)
INTEGER INCX, INCY, N
COMPLEX*16 A
COMPLEX*16 X(*), Y(*)

Description
The SAXPY, DAXPY, CAXPY, or ZAXPY subroutine computes a constant times a vector plus a vector:

\[ Y = A \cdot X + Y \]

Parameters

\( N \) On entry, \( N \) specifies the number of elements in \( X \) and \( Y \); unchanged on exit.
\( A \) On entry, \( A \) contains a constant to be multiplied by the \( X \) vector; unchanged on exit.
\( X \) Vector of dimension at least \( (1 + (N-1) \cdot \text{abs(INCX)} \) ); unchanged on exit.
\( \text{INCX} \) On entry, \( \text{INCX} \) specifies the increment for the elements of \( X \); unchanged on exit.
\( Y \) Vector of dimension at least \( (1 + (N-1) \cdot \text{abs(INCY)} \) ); the result is returned in vector \( Y \).
\( \text{INCY} \) On entry, \( \text{INCY} \) specifies the increment for the elements of \( Y \); unchanged on exit.

Error Codes
If \( SA = 0 \) or \( N <= 0 \), the subroutine returns immediately.

SROTG, DROTG, CROTG, or ZROTG Subroutine

Purpose
Constructs Givens plane rotation.

Library
BLAS Library (libblas.a)
FORTRAN Syntax

SUBROUTINE SROTG(A,B,C,S)
REAL A, B, C, S
SUBROUTINE DROTG(A,B,C,S)
DOUBLE PRECISION A,B,C,S
SUBROUTINE CROTG(A,B,C,S)
REAL C
COMPLEX A,B,S
SUBROUTINE ZROTG(A,B,C,S)
DOUBLE PRECISION C
COMPLEX*16 A,B,S

Description
Given vectors $A$ and $B$, the SROTG, DROTG, CROTG, or ZROTG subroutine computes:

\[
\begin{align*}
A & = \frac{a}{|A| + |B|} \\
B & = \frac{b}{|A| + |B|} \\
\end{align*}
\]

\[
\text{roe} = \begin{cases} 
A & \text{if } |A| > |B| \\
B & \text{if } |B| \geq |A| 
\end{cases}
\]

\[
C = \begin{cases} 
A/r & \text{if } r \neq 0 \} \\
1 & \text{if } r = 0 \} 
\end{cases}
\]

\[
S = \begin{cases} 
B/r & \text{if } r \neq 0 \} \\
0 & \text{if } r = 0 \} 
\end{cases}
\]

The numbers $C$, $S$, and $r$ then satisfy the matrix equation:

\[
\begin{bmatrix}
C & S \\
-S & C
\end{bmatrix}
\begin{bmatrix}
A \\
B
\end{bmatrix} =
\begin{bmatrix}
r \\
0
\end{bmatrix}
\]

The subroutines also compute:

\[
z = \begin{cases} 
1/C & \text{if } |B| \geq |A| \text{ and } C \neq 0, \\
1 & \text{if } C = 0.
\end{cases}
\]

The subroutines return $r$ overwriting $A$ and $z$ overwriting $B$, as well as returning $C$ and $S$.

Parameters

$A$ On entry, contains a scalar constant; on exit, contains the value $r$.
$B$ On entry, contains a scalar constant; on exit, contains the value $z$.
$C$ Can contain any value on entry; the value $C$ returned on exit.
$S$ Can contain any value on entry; the value $S$ returned on exit.

SROT, DROT, CSROT, or ZDROT Subroutine

Purpose
Applies a plane rotation.

Library
BLAS Library (libblas.a)
FORTRAN Syntax

SUBROUTINE SROT(N, X, INCX, Y, INCY, C, S)
INTEGER INCX, INCY, N
REAL C, S
REAL X(*), Y(*)
SUBROUTINE DROT(N, X, INCX, Y, INCY, C, S)
INTEGER INCX, INCY, N
DOUBLE PRECISION C, S
DOUBLE PRECISION X(*), Y(*)
SUBROUTINE CSROT(N, X, INCX, Y, INCY, C, S)
INTEGER INCX, INCY, N
REAL C, S
COMPLEX X(*), Y(*)
SUBROUTINE ZDROT(N, X, INCX, Y, INCY, C, S)
INTEGER INCX, INCY, N
DOUBLE PRECISION C, S
COMPLEX*16 X(*), Y(*)

Description
The SROT, DROT, CSROT, or ZDROT subroutine computes:

\[ \begin{array}{c|c|c}
    X & C & S \\
    \hline
    Y & -S & C \\
\end{array} \]

for \( i = 1, \ldots, N \).

The subroutines return the modified \( X \) and \( Y \).

Parameters

\( N \) On entry, \( N \) specifies the number of elements in \( X \) and \( Y \); unchanged on exit.
\( X \) Vector of dimension at least \((1 + (N-1) \times \text{abs(INCX)})\); unchanged on exit.
\( INCX \) On entry, \( INCX \) specifies the increment for the elements of \( X \); unchanged on exit.
\( Y \) Vector of dimension at least \((1 + (N-1) \times \text{abs(INCY)})\); modified on exit.
\( INCY \) On entry, \( INCY \) specifies the increment for the elements of \( Y \); unchanged on exit.
\( C \) Scalar constant; unchanged on exit.
\( S \) Scalar constant; unchanged on exit.

Error Codes
If \( N \leq 0 \), or if \( C = 1 \) and \( S = 0 \), the subroutines return immediately.

SCOPY, DCOPY, CCOPY, or ZCOPY Subroutine

Purpose
Copies vector \( X \) to \( Y \).

Library
BLAS Library (libblas.a)
FORTRAN Syntax

SUBROUTINE SCOPY(N, X, INCX, Y, INCY)
INTEGER INCX, INCY, N
REAL X(*), Y(*)
SUBROUTINE DCOPY(N, X, INCX, Y, INCY)
INTEGER INCX, INCY, N
DOUBLE PRECISION X(*), Y(*)
SUBROUTINE CCOPY(N, X, INCX, Y, INCY)
INTEGER INCX, INCY, N
COMPLEX X(*), Y(*)
SUBROUTINE ZCOPY(N, X, INCX, Y, INCY)
INTEGER INCX, INCY, N
COMPLEX*16 X(*), Y(*)

Description
The SCOPY, DCOPY, CCOPY, or ZCOPY subroutine copies vector X to vector Y.

Parameters
N On entry, N specifies the number of elements in X and Y; unchanged on exit.
X Vector of dimension at least (1 + (N-1) * abs(INCX)); unchanged on exit.
INCX On entry, INCX specifies the increment for the elements of X; unchanged on exit.
Y Vector of dimension at least (1 + (N-1) * abs(INCY)) or greater; can contain any values on entry; on exit, contains the same values as X.
INCY On entry, INCY specifies the increment for the elements of Y; unchanged on exit.

Error Codes
For values of N <= 0, the subroutines return immediately.

SSWAP, DSWAP, CSWAP, or ZSWAP Subroutine

Purpose
Interchanges vectors X and Y.

Library
BLAS Library (libblas.a)

FORTRAN Syntax

SUBROUTINE SSWAP(N, X, INCX, Y, INCY)
INTEGER INCX, INCY, N
REAL X(*), Y(*)
SUBROUTINE DSWAP(N, X, INCX, Y, INCY)
INTEGER INCX, INCY, N
DOUBLE PRECISION X(*), Y(*)
SUBROUTINE CSWAP(N, X, INCX, Y, INCY)
INTEGER INCX, INCY, N
COMPLEX X(*), Y(*)
SUBROUTINE ZSWAP(N, X, INCX, Y, INCY)
INTEGER INCX, INCY, N
COMPLEX*16 X(*), Y(*)

716 Technical Reference, Volume 2: Base Operating System and Extensions
Description
The SSWAP, DSWAP, CSWAP, or ZSWAP subroutine interchanges vector $X$ and vector $Y$.

Parameters

- $N$ On entry, $N$ specifies the number of elements in $X$ and $Y$; unchanged on exit.
- $X$ Vector of dimension at least $(1 + (N-1) * \text{abs(INCX)})$; on exit, contains the elements of vector $Y$.
- INCX On entry, INCX specifies the increment for the elements of $X$; unchanged on exit.
- $Y$ Vector of dimension at least $(1 + (N-1) * \text{abs(INCY)})$; on exit, contains the elements of vector $X$.
- INCY On entry, INCY specifies the increment for the elements of $Y$; unchanged on exit.

Error Codes
For values of $N \leq 0$, the subroutines return immediately.

SNRM2, DNRM2, SCNRM2, or DZNRM2 Function

Purpose
Computes the Euclidean length of the $N$-vector stored in $X()$ with storage increment INCX.

Library
BLAS Library (libblas.a)

FORTRAN Syntax

REAL FUNCTION SNRM2($N$, $X$, INCX)
INTEGER [INX, N]
REAL $X(*)$

DOUBLE PRECISION FUNCTION DNRM2($N$, $X$, INCX)
INTEGER $N$
DOUBLE PRECISION $X(*)$

REAL FUNCTION SCNRM2($N$, $X$, INCX)
INTEGER $N$
COMPLEX $X(*)$

DOUBLE PRECISION FUNCTION DZNRM2($N$, $X$, INCX)
INTEGER $N$
COMPLEX*16 $X(*)$

Description
The SNRM2, DNRM2, SCNRM2, or DZNRM2 function returns the Euclidean norm of the $N$-vector stored in $X()$ with storage increment INCX.

Parameters

- $N$ On entry, $N$ specifies the number of elements in $X$ and $Y$; unchanged on exit.
- $X$ Vector of dimension at least $(1 + (N-1) * \text{abs(INCX)})$; unchanged on exit.
- INCX On entry, INCX specifies the increment for the elements of $X$; INCX must be greater than 0; unchanged on exit.

Error Codes
For values of $N \leq 0$, a value of 0 is returned.
SASUM, DASUM, SCASUM, or DZASUM Function

Purpose

Returns the sum of absolute values of vector components.

Library

BLAS Library (libblas.a)

FORTRAN Syntax

```fortran
REAL FUNCTION SASUM(N,X,INCX)
INTEGER INCX,N
REAL X(*)

DOUBLE PRECISION FUNCTION DASUM(N,X,INCX)
INTEGER INCX,N
DOUBLE PRECISION X(*)

REAL FUNCTION SCASUM(N,X,INCX)
INTEGER INCX,N
COMPLEX X(*)

DOUBLE PRECISION FUNCTION DZASUM(N,X,INCX)
INTEGER INCX,N
COMPLEX*16 X(*)
```

Description

The SASUM, DASUM, SCASUM, or DZASUM function returns the sum of absolute values of vector components.

Parameters

- **N**
  On entry, \( N \) specifies the number of elements in \( X \) and \( Y \); unchanged on exit.
- **X**
  Vector of dimension at least \( 1 + (N-1) \times \text{abs(INCX)} \); unchanged on exit.
- **INCX**
  On entry, \( INCX \) specifies the increment for the elements of \( X \); \( INCX \) must be greater than 0; unchanged on exit.

Error Codes

For values of \( N \leq 0 \), a value of 0 is returned.

SSCAL, DSCAL, CSSCAL, CSCAL, ZDSCAL, or ZSCAL Subroutine

Purpose

Scales a vector by a constant.

Library

BLAS Library (libblas.a)
FORTRAN Syntax

SUBROUTINE SSCALE(N, A, X, INCX)
INTEGER INCX, N
REAL A
REAL X(*)
SUBROUTINE DSCALE(N, A, X, INCX)
INTEGER INCX, N
DOUBLE PRECISION A
DOUBLE PRECISION X(*)
SUBROUTINE CSCALE(N, A, X, INCX)
INTEGER INCX, N
REAL A
COMPLEX X(*)
SUBROUTINE ZSCALE(N, A, X, INCX)
INTEGER INCX, N
COMPLEX A
COMPLEX X(*)

Description
The SSCALE, DSCALE, CSSCALE, CSCAL, ZDSCALE, or ZSCALE subroutine scales a vector by a constant:

\[ X := X \ast A \]

Parameters

- **N**
  - On entry, \( N \) specifies the number of elements in \( X \) and \( Y \); unchanged on exit.
- **A**
  - Scaling constant; unchanged on exit.
- **X**
  - Vector of dimension at least \((1 + (N-1) \ast \text{abs(INCX)})\); on exit, contains the scaled vector.
- **INCX**
  - On entry, \( INCX \) specifies the increment for the elements of \( X \); \( INCX \) must be greater than 0; unchanged on exit.

Error Codes
For values of \( N \leq 0 \), the subroutines return immediately.

ISAMAX, IDAMAX, ICAMAX, or IZAMAX Function

Purpose
Finds the index of element having maximum absolute value.

Library
BLAS Library (libblas.a)
FORTRAN Syntax

```
INTEGER FUNCTION ISAMAX(N,X,INCX)
INTEGER INCX, N
REAL X(*)
INTEGER FUNCTION IDAMAX(N,X,INCX)
INTEGER INCX, N
DOUBLE PRECISION X(*)
INTEGER FUNCTION ICAMAX(N,X,INCX)
INTEGER INCX, N
COMPLEX X(*)
INTEGER FUNCTION IZAMAX(N,X,INCX)
INTEGER INCX, N
COMPLEX*16 X(*)
```

Description
The ISAMAX, IDAMAX, ICAMAX, or IZAMAX function returns the index of element having maximum absolute value.

Parameters

- **N** On entry, N specifies the number of elements in X and Y; unchanged on exit.
- **X** Vector of dimension at least \((1 + (N-1) \times \text{abs(INCX)})\); unchanged on exit.
- **INCX** On entry, INCX specifies the increment for the elements of X; unchanged on exit.

Error Codes
For values of \(N <= 0\), a value of 0 is returned.

SDSDOT Function

Purpose
Returns the dot product of two vectors plus a constant.

Library
BLAS Library (libblas.a)

FORTRAN Syntax

```
REAL FUNCTION SDSDOT(N,B,X,INCX,Y,INCY)
INTEGER INCX, INCY
REAL B, X(*), Y(*)
```

Purpose
The SDSDOT function computes the sum of constant B and dot product of vectors X and Y.

Note: Computation is performed in double precision.

Parameters

- **N** On entry, N specifies the number of elements in X and Y; unchanged on exit.
- **B** Scalar; unchanged on exit.
- **X** Vector of dimension at least \((1 + (N-1) \times \text{abs(INCX)})\); unchanged on exit.
**INCX** On entry, INCX specifies the increment for the elements of X; INCX must be greater than zero; unchanged on exit.

**Y** Vector of dimension at least \((1 + (N-1) \times \text{abs(INCY)})\); unchanged on exit.

**INCY** On entry, INCY specifies the increment for the elements of Y; INCY must be greater than 0; unchanged on exit.

### Error Codes
For values of \(N \leq 0\), the subroutine returns immediately.

---

### SROTM or DROTM Subroutine

#### Purpose
Applies the modified Givens transformation.

#### Library
BLAS Library (libblas.a)

#### FORTRAN Syntax

```fortran
SUBROUTINE SROTM(N, X, INCX, Y, INCY, PARAM)
INTEGER N, INCX, INCY
REAL X(*), Y(*), PARAM(5)

SUBROUTINE DROTM(N, X, INCX, Y, INCY, PARAM)
INTEGER N, INCX, INCY
DOUBLE PRECISION X(*), Y(*), PARAM(5)
```

#### Description
Let \(H\) denote the modified Givens transformation defined by the parameter array \(PARAM\). The *SROTM* or *DROTM* subroutine computes:

\[
\begin{pmatrix}
  x \\ y
\end{pmatrix}
:=
\begin{pmatrix}
  H(1,1) & H(1,2) \\
  H(2,1) & H(2,2)
\end{pmatrix}
\begin{pmatrix}
  x \\ y
\end{pmatrix}
\]

where \(H\) is a 2 x 2 matrix with the components defined by the elements of the array \(PARAM\) as follows:

- if \(PARAM(1) == 0.0\)
  - \(H(1,1) = H(2,2) = 1.0\)
  - \(H(2,1) = PARAM(3)\)
  - \(H(1,2) = PARAM(4)\)
- if \(PARAM(1) == 1.0\)
  - \(H(1,2) = H(2,1) = -1.0\)
  - \(H(1,1) = PARAM(2)\)
  - \(H(2,2) = PARAM(5)\)
- if \(PARAM(1) == -1.0\)
  - \(H(1,1) = PARAM(2)\)
  - \(H(2,1) = PARAM(3)\)
  - \(H(1,2) = PARAM(4)\)
  - \(H(2,2) = PARAM(5)\)
- if \(PARAM(1) == -2.0\)
  - \(H = I\) (Identity matrix)

If \(N \leq 0\) or \(H\) is an identity matrix, the subroutines return immediately.
Parameters

\( N \)  
On entry, \( N \) specifies the number of elements in \( X \) and \( Y \); unchanged on exit.

\( X \)  
Vector of dimension at least \( (1 + (N-1) \times \text{abs(INCX)}) \); on exit, modified as described above.

\( INCX \)  
On entry, \( INCX \) specifies the increment for the elements of \( X \); \( INCX \) must be greater than 0; unchanged on exit.

\( Y \)  
Vector of dimension at least \( (1 + (N-1) \times \text{abs(INCY)}) \); on exit, modified as described above.

\( INCY \)  
On entry, \( INCY \) specifies the increment for the elements of \( Y \); \( INCY \) must be greater than 0; unchanged on exit.

\( PARAM \)  
Vector of dimension (5); on entry, must be set as described above. Specifically, \( PARAM(1) \) is a flag and must have value of either 0.0, -1.0, 1.0, or 2.0; unchanged on exit.

Related information

The SROTMG or DROTMG subroutine builds the \( PARAM \) array prior to use by the SROT or DROT subroutine.

SROTMG or DROTMG Subroutine

Purpose

Constructs a modified Givens transformation.

Library

BLAS Library (libblas.a)

FORTRAN Syntax

SUBROUTINE SROTMG(D1, D2, X1, X2, PARAM)  
REAL D1, D2, X1, X2, PARAM(5)  
SUBROUTINE DROTMG(D1, D2, X1, X2, PARAM)  
DOUBLE PRECISION D1, D2, X1, X2, PARAM(5)

Description

The SROTMG or DROTMG subroutine constructs a modified Givens transformation. The input quantities \( D1, D2, X1, \) and \( X2 \) define a 2-vector in partitioned form:

\[
\begin{bmatrix}
a1 & a2 \\
\end{bmatrix}
\begin{bmatrix}
\text{\text{sqrt(D1)}} & 0 \\
0 & \text{\text{sqrt(D2)}} \\
\end{bmatrix}
\begin{bmatrix}
X1 \\
X2 \\
\end{bmatrix}
\]

The subroutines determine the modified Givens rotation matrix \( H \) that transforms \( X2 \) and, thus, \( a2 \) to 0. A representation of this matrix is stored in the array \( PARAM \) as follows:

Case 1: \( PARAM(1) = 1.0 \)
- \( PARAM(2) = H(1,1) \)
- \( PARAM(5) = H(2,2) \)

Case 2: \( PARAM(1) = 0.0 \)
- \( PARAM(3) = H(2,1) \)
- \( PARAM(4) = H(1,2) \)

Case 3: \( PARAM(1) = -1.0 \)
- \( H(1,1) = PARAM(2) \)
- \( H(2,1) = PARAM(3) \)
- \( H(1,2) = PARAM(4) \)
- \( H(2,2) = PARAM(5) \)
Case 4: \( \text{PARAM}(1) = -2.0 \)
\( \text{H} = \text{I} \) (Identity matrix)

**Note:** Locations in \text{PARAM} not listed are left unchanged.

**Parameters**

- \( D1 \) Nonnegative scalar; modified on exit to reflect the results of the transformation.
- \( D2 \) Scalar; can be negative on entry; modified on exit to reflect the results of the transformation.
- \( X1 \) Scalar; modified on exit to reflect the results of the transformation.
- \( X2 \) Scalar; unchanged on exit.
- \( \text{PARAM} \) Vector of dimension (5); values on entry are unused; modified on exit as described above.

**Related Information**

The \text{SROTM} and \text{DROTM} ("SROTM or DROTM Subroutine" on page 721) subroutines apply the Modified Givens Transformation.

**SGEMV, DGEMV, CGEMV, or ZGEMV Subroutine**

**Purpose**

Performs matrix-vector operation with general matrices.

**Library**

BLAS Library (libblas.a)

**FORTRAN Syntax**

SUBROUTINE SGEMV(\( \text{TRANS}, M, N, \text{ALPHA}, A, \text{LDA}, X, \text{INCX}, \text{BETA}, Y, \text{INCY} \))
REAL \( \text{ALPHA}, \text{BETA} \)
INTEGER \( \text{INCX}, \text{INCY}, \text{LDA}, M, N \)
CHARACTER*1 \( \text{TRANS} \)
REAL \( A\text{(LDA,*), X(*), Y(*)} \)

SUBROUTINE DGEMV(\( \text{TRANS}, M, N, \text{ALPHA}, A, \text{LDA}, X, \text{INCX}, \text{BETA}, Y, \text{INCY} \))
DOUBLE PRECISION \( \text{ALPHA,BETA} \)
INTEGER \( \text{INCX}, \text{INCY}, \text{LDA}, M, N \)
CHARACTER*1 \( \text{TRANS} \)
DOUBLE PRECISION \( A\text{(LDA,*), X(*), Y(*)} \)

SUBROUTINE CGEMV(\( \text{TRANS}, M, N, \text{ALPHA}, A, \text{LDA}, X, \text{INCX}, \text{BETA}, Y, \text{INCY} \))
COMPLEX \( \text{ALPHA,BETA} \)
INTEGER \( \text{INCX}, \text{INCY}, \text{LDA}, M, N \)
CHARACTER*1 \( \text{TRANS} \)
COMPLEX \( A\text{(LDA,*), X(*), Y(*)} \)

SUBROUTINE ZGEMV(\( \text{TRANS}, M, N, \text{ALPHA}, A, \text{LDA}, X, \text{INCX}, \text{BETA}, Y, \text{INCY} \))
COMPLEX*16 \( \text{ALPHA,BETA} \)
INTEGER \( \text{INCX}, \text{INCY}, \text{LDA}, M, N \)
CHARACTER*1 \( \text{TRANS} \)
COMPLEX*16 \( A\text{(LDA,*), X(*), Y(*)} \)
The \texttt{SGEMV}, \texttt{DGEMV}, \texttt{CGEMV}, or \texttt{ZGEMV} subroutine performs one of the following matrix-vector operations:

\begin{align*}
y := & \alpha \times A \times x + \beta \times y \\
\text{OR} \\
y := & \alpha \times A' \times x + \beta \times y
\end{align*}

where alpha and beta are scalars, x and y are vectors, and A is an $M \times N$ matrix.

\textbf{Parameters}

\begin{tabular}{ll}
\textit{TRANS} & On entry, \textit{TRANS} specifies the operation to be performed as follows: \\
\textit{TRANS} = 'N' or 'n' & \\
y := & \alpha \times A \times x + \beta \times y \\
\textit{TRANS} = 'T' or 't' & \\
y := & \alpha \times A' \times x + \beta \times y \\
\textit{TRANS} = 'C' or 'c' & \\
y := & \alpha \times A' \times x + \beta \times y \\
\end{tabular}

Unchanged on exit.

\begin{tabular}{ll}
\textit{M} & On entry, \textit{M} specifies the number of rows of the matrix A; \textit{M} must be at least 0; unchanged on exit. \\
\textit{N} & On entry, \textit{N} specifies the number of columns of the matrix A; \textit{N} must be at least 0; unchanged on exit. \\
\textit{ALPHA} & On entry, \textit{ALPHA} specifies the scalar alpha; unchanged on exit. \\
\textit{A} & An array of dimension ( \textit{LDA}, \textit{N} ); on entry, the leading \textit{M} by \textit{N} part of the array \textit{A} must contain the matrix of coefficients; unchanged on exit. \\
\textit{LDA} & On entry, \textit{LDA} specifies the first dimension of \textit{A} as declared in the calling (sub) program; \textit{LDA} must be at least \text{max}( 1, \textit{M} ); unchanged on exit. \\
\textit{X} & A vector of dimension at least \text{1 + (N-1) * abs( \textit{INCX} )} when \textit{TRANS} = 'N' or 'n', otherwise, at least \text{1 + (M-1) * abs( \textit{INCX} )}; on entry, the incremented array \textit{X} must contain the vector \textit{x}; unchanged on exit. \\
\textit{INCY} & On entry, \textit{INCX} specifies the increment for the elements of \textit{X}; \textit{INCY} must not be 0; unchanged on exit. \\
\textit{BETA} & On entry, \textit{BETA} specifies the scalar beta; when \textit{BETA} is supplied as 0, \textit{Y} need not be set on input; unchanged on exit. \\
\textit{Y} & A vector of dimension at least \text{1 + (M-1) * abs( \textit{INCY} )} when \textit{TRANS} = 'N' or 'n', otherwise at least \text{1 + (N-1) * abs( \textit{INCY} )}; on entry, with \textit{BETA} nonzero, the incremented array \textit{Y} must contain the vector \textit{y}; on exit, \textit{Y} is overwritten by the updated vector \textit{y}. \\
\textit{INCY} & On entry, \textit{INCY} specifies the increment for the elements of \textit{Y}; \textit{INCY} must not be 0; unchanged on exit.
\end{tabular}

\textbf{SGBMV, DGBMV, CGBMV, or ZGBMV Subroutine}

\textbf{Purpose}

Performs matrix-vector operations with general banded matrices.

\textbf{Library}

BLAS Library (libblas.a)

\textbf{FORTRAN Syntax}

\begin{verbatim}
SUBROUTINE SGBMV(TRANS, M, N, KL, KU, ALPHA, A, LDA,
X, INCX, BETA, Y, INCY)
REAL   ALPHA, BETA
\end{verbatim}
INTEGER INCX, INCY, KL, KU, LDA, M, N
CHARACTER*1 TRANS
REAL A(LDA,*), X(*), Y(*)
SUBROUTINE DGBMV(TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
DOUBLE PRECISION ALPHA, BETA
INTEGER INCX, INCY, KL, KU, LDA, M, N
CHARACTER*1 TRANS
DOUBLE PRECISION A(LDA,*), X(*), Y(*)
SUBROUTINE CGBMV(TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
COMPLEX ALPHA, BETA
INTEGER INCX, INCY, KL, KU, LDA, M, N
CHARACTER*1 TRANS
COMPLEX A(LDA,*), X(*), Y(*)
SUBROUTINE ZGBMV(TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
COMPLEX*16 ALPHA, BETA
INTEGER INCX, INCY, KL, KU, LDA, M, N
CHARACTER*1 TRANS
COMPLEX*16 A(LDA,*), X(*), Y(*)

Description
The SGBMV, DGBMV, CGBMV, or ZGBMV subroutine performs one of the following matrix-vector operations:

\[ y := \alpha \times A \times x + \beta \times y \]

OR

\[ y := \alpha \times A' \times x + \beta \times y \]

where \( \alpha \) and \( \beta \) are scalars, \( x \) and \( y \) are vectors and \( A \) is an \( M \) by \( N \) band matrix, with \( KL \) subdiagonals and \( KU \) superdiagonals.

Parameters

- **TRANS** On entry, **TRANS** specifies the operation to be performed as follows:
  - **TRANS** = 'N' or 'n'
    \[ y := \alpha \times A \times x + \beta \times y \]
  - **TRANS** = 'T' or 't'
    \[ y := \alpha \times A' \times x + \beta \times y \]
  - **TRANS** = 'C' or 'c'
    \[ y := \alpha \times A' \times x + \beta \times y \]

  Unchanged on exit.

- **M** On entry, **M** specifies the number of rows of the matrix \( A \); **M** must be at least 0; unchanged on exit.

- **N** On entry, **N** specifies the number of columns of the matrix \( A \); **N** must be at least 0; unchanged on exit.

- **KL** On entry, **KL** specifies the number of subdiagonals of the matrix \( A \); **KL** must satisfy 0 .le. \( KL \); unchanged on exit.

- **KU** On entry, **KU** specifies the number of superdiagonals of the matrix \( A \); **KU** must satisfy 0 .le. \( KU \); unchanged on exit.

- **ALPHA** On entry, **ALPHA** specifies the scalar \( \alpha \); unchanged on exit.
A vector of dimension \((LDA, N)\); on entry, the leading \((KL + KU + 1)\) by \(N\) part of the array \(A\) must contain the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row \((KU + 1)\) of the array, the first superdiagonal starting at position 2 in row \(KU\), the first subdiagonal starting at position 1 in row \((KU + 2)\), and so on. Elements in the array \(A\) that do not correspond to elements in the band matrix (such as the top left \(KU\) by \(KU\) triangle) are not referenced. The following program segment transfers a band matrix from conventional full matrix storage to band storage:

\[
\text{DO } 20, J = 1, N \\
\quad K = KU + 1 - J \\
\text{DO } 10, I = \max(1, J - KU), \min( M, J + KL ) \\
\quad A(K + I, J) = \text{matrix}(I, J) \\
10 \text{ CONTINUE} \\
20 \text{ CONTINUE}
\]

Unchanged on exit.

\(LDA\) On entry, \(LDA\) specifies the first dimension of \(A\) as declared in the calling (sub) program. \(LDA\) must be at least \((KL + KU + 1)\); unchanged on exit.

\(X\) A vector of dimension at least \((1 + (N-1) \times \text{abs(INCX)})\) when \(TRANS = 'N'\) or \('n'\), otherwise, at least \((1 + (M-1) \times \text{abs(INCX)})\); on entry, the incremented array \(X\) must contain the vector \(x\); unchanged on exit.

\(INCY\) On entry, \(INCY\) specifies the increment for the elements of \(X\); \(INCY\) must not be 0; unchanged on exit.

\(BETA\) On entry, \(BETA\) specifies the scalar \(\beta\); when \(BETA\) is supplied as 0 then \(Y\) need not be set on input; unchanged on exit.

\(Y\) A vector of dimension at least \((1 + (M-1) \times \text{abs(INCY)})\) when \(TRANS = 'N'\) or \('n'\), otherwise, at least \((1 + (N-1) \times \text{abs(INCY)})\); on entry, the incremented array \(Y\) must contain the vector \(y\); on exit, \(Y\) is overwritten by the updated vector \(y\).

\(INCY\) On entry, \(INCY\) specifies the increment for the elements of \(Y\); \(INCY\) must not be 0; unchanged on exit.

---

### CHEMV or ZHEMV Subroutine

#### Purpose
Performs matrix-vector operations using Hermitian matrices.

#### Library
BLAS Library \((\text{libblas.a})\)

#### FORTRAN Syntax

| SUBROUTINE CHEMV \((UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)\) |
|-------------------------|-----------------|-----------------|
| \( \text{COMPLEX} \)  | \( \text{ALPHA} \) | \( \text{BETA} \) |
| \( \text{INTEGER} \)  | \( \text{INCY} \) | \( \text{INCY} \) | \( \text{LDA} \) | \( \text{N} \) |
| \( \text{CHARACTER}^1 \) | \( \text{UPLO} \) |
| \( \text{COMPLEX}^*1 \) | \( A(LDA,*) \), \( X(*) \), \( Y(*) \) |

| SUBROUTINE ZHEMV \((UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)\) |
|-------------------------|-----------------|-----------------|
| \( \text{COMPLEX}^*16 \) | \( \text{ALPHA,BETA} \) |
| \( \text{INTEGER} \)  | \( \text{INCY},\text{LDA},\text{N} \) |
| \( \text{CHARACTER}^*1 \) | \( \text{UPLO} \) |
| \( \text{COMPLEX}^*16 \) | \( A(LDA,*) \), \( X(*) \), \( Y(*) \) |

#### Description
The CHEMV or ZHEMV subroutine performs the matrix-vector operation:

\[
y := \alpha \times A \times x + \beta \times y
\]

where \(\alpha\) and \(\beta\) are scalars, \(x\) and \(y\) are \(N\) element vectors and \(A\) is an \(N\) by \(N\) Hermitian matrix.
Parameters

**UPLO**
On entry, **UPLO** specifies whether the upper or lower triangular part of the array **A** is to be referenced as follows:

- \( **UPLO = 'U' \) or \( 'u' \)
  - Only the upper triangular part of **A** is to be referenced; unchanged on exit.
- \( **UPLO = 'L' \) or \( 'l' \)
  - Only the lower triangular part of **A** is to be referenced; unchanged on exit.

**N**
On entry, **N** specifies the order of the matrix **A**; **N** must be at least 0; unchanged on exit.

**ALPHA**
On entry, **ALPHA** specifies the scalar alpha; unchanged on exit.

**A**
An array of dimension ( **LDA**, **N** ); on entry with **UPLO** = 'U' or 'u', the leading **N** by **N** upper triangular part of the array **A** must contain the upper triangular part of the Hermitian matrix and the strictly lower triangular part of **A** is not referenced; on entry with **UPLO** = 'L' or 'l', the leading **N** by **N** lower triangular part of the array **A** must contain the lower triangular part of the Hermitian matrix and the strictly upper triangular part of **A** is not referenced. The imaginary parts of the diagonal elements need not be set and are assumed to be 0; unchanged on exit.

**LDA**
On entry, **LDA** specifies the first dimension of **A** as declared in the calling (sub) program; **LDA** must be at least max( 1, **N** ); unchanged on exit.

**X**
A vector of dimension at least \( (1 + (N-1) \times \text{abs( INCX )}) \); on entry, the incremented array **X** must contain the **N** element vector **x**; unchanged on exit.

**INCX**
On entry, **INCX** specifies the increment for the elements of **X**; **INCX** must not be 0; unchanged on exit.

**BETA**
On entry, **BETA** specifies the scalar beta; when **BETA** is supplied as 0 then **Y** need not be set on input; unchanged on exit.

**Y**
A vector of dimension at least \( (1 + (N-1) \times \text{abs( INCY )}) \); on entry, the incremented array **Y** must contain the **N** element vector **y**; on exit, **Y** is overwritten by the updated vector **y**.

**INCY**
On entry, **INCY** specifies the increment for the elements of **Y**; **INCY** must not be 0; unchanged on exit.

---

**CHBMV or ZHBMV Subroutine**

**Purpose**
Performs matrix-vector operations using a Hermitian band matrix.

**Library**
BLAS Library (libblas.a)

**FORTRAN Syntax**

```fortran
SUBROUTINE CHBMV(UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
COMPLEX ALPHA, BETA
INTEGER INCX, INCY, N, LDA, K
CHARACTER*1 UPLO
COMPLEX A(LDA,*), X(*), Y(*)
SUBROUTINE ZHBMV(UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
COMPLEX*16 ALPHA, BETA
INTEGER INCX, INCY, K, LDA, N
CHARACTER*1 UPLO
COMPLEX*16 A(LDA,*), X(*), Y(*)
```

**Description**
The **CHBMV** or **ZHBMV** subroutine performs the matrix-vector operation:

\[
y := \alpha \ast A \ast x + \beta \ast y
\]
where alpha and beta are scalars, x and y are $N$ element vectors, and $A$ is an $N$ by $N$ Hermitian band matrix with $K$ superdiagonals.

**Parameters**

**UPLO** On entry, **UPLO** specifies whether the upper or lower triangular part of the band matrix $A$ is being supplied as follows:

- **UPLO** = 'U' or 'u'
  - The upper triangular part of $A$ is being supplied.

- **UPLO** = 'L' or 'l'
  - The lower triangular part of $A$ is being supplied.

Unchanged on exit.

**N** On entry, $N$ specifies the order of the matrix $A$; $N$ must be at least 0; unchanged on exit.

**K** On entry, $K$ specifies the number of superdiagonals of the matrix $A$; $K$ must satisfy $0 \leq K$; unchanged on exit.

**ALPHA** On entry, **ALPHA** specifies the scalar alpha; unchanged on exit.

**A** An array of dimension ($LDA$, $N$). On entry with **UPLO** = 'U' or 'u', the leading $(K + 1)$ by $N$ part of the array $A$ must contain the upper triangular band part of the Hermitian matrix, supplied column by column, with the leading diagonal of the matrix in row $(K + 1)$ of the array, the first superdiagonal starting at position 2 in row $K$, and so on. The top left $K$ by $K$ triangle of the array $A$ is not referenced. The following program segment transfers the upper triangular part of a Hermitian band matrix from conventional full matrix storage to band storage:

```
DO 20, J = 1, N
  M = K + 1 - J
  DO 10, I = MAX( 1, J - K ), J
    A( M + I, J ) = matrix( I, J )
  10 CONTINUE
20 CONTINUE
```

**Note:** On entry with **UPLO** = 'L' or 'l', the leading $(K + 1)$ by $N$ part of the array $A$ must contain the lower triangular band part of the Hermitian matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first subdiagonal starting at position 1 in row 2, and so on. The bottom right $K$ by $K$ triangle of the array $A$ is not referenced. The following program segment transfers the lower triangular part of a Hermitian band matrix from conventional full matrix storage to band storage:

```
DO 20, J = 1, N
  M = 1 - J
  DO 10, I = J, MIN( N, J + K )
    A( M + I, J ) = matrix( I, J )
  10 CONTINUE
20 CONTINUE
```

The imaginary parts of the diagonal elements need not be set and are assumed to be 0. Unchanged on exit.

**LDA** On entry, **LDA** specifies the first dimension of $A$ as declared in the calling (sub) program; **LDA** must be at least $(K + 1)$; unchanged on exit.

**X** A vector of dimension at least $(1 + (N-1) \times \text{abs( INCX )})$; on entry, the incremented array $X$ must contain the vector $x$; unchanged on exit.

**INCY** On entry, **INCY** specifies the increment for the elements of $X$; **INCY** must not be 0; unchanged on exit.

**BETA** On entry, **BETA** specifies the scalar beta unchanged on exit.

**Y** A vector of dimension at least $(1 + (N-1) \times \text{abs( INCY )})$; on entry, the incremented array $Y$ must contain the vector $y$; on exit, $Y$ is overwritten by the updated vector $y$.

**INCY** On entry, **INCY** specifies the increment for the elements of $Y$; **INCY** must not be 0; unchanged on exit.
**CHPMV or ZHPMV Subroutine**

**Purpose**
Performs matrix-vector operations using a packed Hermitian matrix.

**Library**
BLAS Library (`libblas.a`)

**FORTRAN Syntax**

```
SUBROUTINE CHPMV(UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY)
COMPLEX ALPHA, BETA
INTEGER INCX, INCY, N
CHARACTER*1 UPLO
COMPLEX AP(*), X(*), Y(*)
```

```
SUBROUTINE ZHPMV
COMPLEX*16 ALPHA, BETA
INTEGER INCX, INCY, N
CHARACTER*1 UPLO
COMPLEX*16 AP(*), X(*), Y(*)
```

**Description**
The CHPMV or ZHPMV subroutine performs the matrix-vector operation:

\[ y := \alpha \cdot A \cdot x + \beta \cdot y \]

where \( \alpha \) and \( \beta \) are scalars, \( x \) and \( y \) are \( N \) element vectors and \( A \) is an \( N \) by \( N \) Hermitian matrix, supplied in packed form.

**Parameters**

- **UPLO**
  On entry, \( \text{UPLO} \) specifies whether the upper or lower triangular part of the matrix \( A \) is supplied in the packed array \( AP \) as follows:
  
  \( \text{UPLO} = 'U' \) or \( 'u' \)
  - The upper triangular part of \( A \) is supplied in \( AP \).
  
  \( \text{UPLO} = 'L' \) or \( 'l' \)
  - The lower triangular part of \( A \) is supplied in \( AP \).

  Unchanged on exit.

- **N**
  On entry, \( N \) specifies the order of the matrix \( A \); \( N \) must be at least 0; unchanged on exit.

- **ALPHA**
  On entry, \( \text{ALPHA} \) specifies the scalar \( \alpha \); unchanged on exit.

- **AP**
  A vector of dimension at least \((N \times (N+1))/2\); on entry with \( \text{UPLO} = 'U' \) or \( 'u' \), the array \( AP \) must contain the upper triangular part of the Hermitian matrix packed sequentially, column by column, so that \( AP(1) \) contains \( A(1,1) \), \( AP(2) \) and \( AP(3) \) contain \( A(1,2) \) and \( A(2,2) \) respectively, and so on; on entry with \( \text{UPLO} = 'L' \) or \( 'l' \), the array \( AP \) must contain the lower triangular part of the Hermitian matrix packed sequentially, column by column, so that \( AP(1) \) contains \( A(1,1) \), \( AP(2) \) and \( AP(3) \) contain \( A(2,1) \) and \( A(3,1) \) respectively, and so on. The imaginary parts of the diagonal elements need not be set and are assumed to be 0; unchanged on exit.

- **X**
  A vector of dimension at least \((1 + (N-1) \times \text{abs(INCX)})\); on entry, the incremented array \( X \) must contain the \( N \) element vector \( x \); unchanged on exit.

- **INCX**
  On entry, \( \text{INCX} \) specifies the increment for the elements of \( X \); \( \text{INCX} \) must not be 0; unchanged on exit.

- **BETA**
  On entry, \( \text{BETA} \) specifies the scalar \( \beta \); when \( \text{BETA} \) is supplied as 0 then \( Y \) need not be set on input; unchanged on exit.
A vector of dimension at least \((1 + (N-1) \times \text{abs}(\text{INCY}))\); on entry, the incremented array \(Y\) must contain the \(N\) element vector \(y\); on exit, \(Y\) is overwritten by the updated vector \(y\).

\(\text{INCY}\) On entry, \(\text{INCY}\) specifies the increment for the elements of \(Y\); \(\text{INCY}\) must not be 0; unchanged on exit.

**SSYMV or DSYMV Subroutine**

**Purpose**
Performs matrix-vector operations using a symmetric matrix.

**Library**
BLAS Library (libblas.a)

**FORTRAN Syntax**

```fortran
SUBROUTINE SSYMV(UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
   REAL  ALPHA, BETA
   INTEGER  INCI, INCY, LDA, N
   CHARACTER*1  UPLO
   REAL  A(LDA,*), X(*), Y(*)
SUBROUTINE DSYMV(UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
   DOUBLE PRECISION ALPHA, BETA
   INTEGER  INCI, INCY, LDA, N
   CHARACTER*1  UPLO
   DOUBLE PRECISION A(LDA,*), X(*), Y(*)
```

**Description**
The SSYMV or DSYMV subroutine performs the matrix-vector operation:

\[ y := \text{alpha} \ast A \ast x + \text{beta} \ast y \]

where alpha and beta are scalars, \(x\) and \(y\) are \(N\) element vectors and \(A\) is an \(N\) by \(N\) symmetric matrix.

**Parameters**

- **UPLO** On entry, \(\text{UPLO}\) specifies whether the upper or lower triangular part of the array \(A\) is to be referenced as follows:
  
  - \(\text{UPLO} = 'U'\) or \('u'\)
    - Only the upper triangular part of \(A\) is to be referenced.
  
  - \(\text{UPLO} = 'L'\) or \('l'\)
    - Only the lower triangular part of \(A\) is to be referenced.

  Unchanged on exit.

- **N** On entry, \(N\) specifies the order of the matrix \(A\); \(N\) must be at least 0; unchanged on exit.

- **ALPHA** On entry, \(\text{ALPHA}\) specifies the scalar alpha; unchanged on exit.

- **A** An array of dimension \((\text{LDA}, N)\); on entry with \(\text{UPLO} = 'U'\) or \('u'\), the leading \(N\) by \(N\) upper triangular part of the array \(A\) must contain the upper triangular part of the symmetric matrix; the strictly lower triangular part of \(A\) is not referenced; on entry with \(\text{UPLO} = 'L'\) or \('l'\), the leading \(N\) by \(N\) lower triangular part of the array \(A\) must contain the lower triangular part of the symmetric matrix; the strictly upper triangular part of \(A\) is not referenced; unchanged on exit.

- **LDA** On entry, \(\text{LDA}\) specifies the first dimension of \(A\) as declared in the calling (sub) program; \(\text{LDA}\) must be at least \(\text{max}(1, N)\); unchanged on exit.
A vector of dimension at least $(1 + (N-1) \cdot \text{abs}(INCX))$; on entry, the incremented array $X$ must contain the $N$ element vector $x$; unchanged on exit.

On entry, $INCX$ specifies the increment for the elements of $X$; $INCX$ must not be 0; unchanged on exit.

On entry, $BETA$ specifies the scalar beta; when $BETA$ is supplied as 0 then $Y$ need not be set on input; unchanged on exit.

A vector of dimension at least $(1 + (N-1) \cdot \text{abs}(INCY))$; on entry, the incremented array $Y$ must contain the $N$ element vector $y$; on exit, $Y$ is overwritten by the updated vector $y$.

On entry, $INCY$ specifies the increment for the elements of $Y$; $INCY$ must not be 0; unchanged on exit.

SSBMV or DSBMV Subroutine

Purpose
Performs matrix-vector operations using symmetric band matrix.

Library
BLAS Library (libblas.a)

FORTRAN Syntax

```
SUBROUTINE SSBMV(UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
REAL ALPHA, BETA
INTEGER INCX, INCY, K, LDA, N
CHARACTER*1 UPLO
REAL A(LDA,*), X(*), Y(*)
```

```
SUBROUTINE DSBMV(UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)
DOUBLE PRECISION ALPHA, BETA
INTEGER INCX, INCY, K, LDA, N
CHARACTER*1 UPLO
DOUBLE PRECISION A(LDA,*), X(*), Y(*)
```

Description
The SSBMV or DSBMV subroutine performs the matrix-vector operation:

$$y := \alpha \cdot A \cdot x + \beta \cdot y$$

where alpha and beta are scalars, $x$ and $y$ are $N$ element vectors, and $A$ is an $N$ by $N$ symmetric band matrix with $K$ super-diagonals.

Parameters

- **UPLO** On entry, UPLO specifies whether the upper or lower triangular part of the band matrix $A$ is being supplied as follows:
  
  $UPLO = \text{'}U\text{'}$ or $\text{'u'}$
  
  The upper triangular part of $A$ is being supplied.

  $UPLO = \text{'}L\text{'}$ or $\text{'l'}$
  
  The lower triangular part of $A$ is being supplied.

  Unchanged on exit.

- **N** On entry, $N$ specifies the order of the matrix $A$; $N$ must be at least 0; unchanged on exit.

- **K** On entry, $K$ specifies the number of superdiagonals of the matrix $A$; $K$ must satisfy 0 .le. $K$; unchanged on exit.

- **ALPHA** On entry, ALPHA specifies the scalar alpha; unchanged on exit.
An array of dimension \((LDA, N)\); on entry with \(UPLO = 'U'\) or \('u'\), the leading \((K + 1)\) by \(N\) part of the array \(A\) must contain the upper triangular band part of the symmetric matrix, supplied column by column, with the leading diagonal of the matrix in row \((K + 1)\) of the array, the first superdiagonal starting at position 2 in row \(K\), and so on. The top left \(K\) by \(K\) triangle of the array \(A\) is not referenced. The following program segment transfers the upper triangular part of a symmetric band matrix from conventional full matrix storage to band storage:

\[
\text{DO } 20, J = 1, N \\
\quad M = K + 1 - J \\
\quad \text{DO } 10, I = \text{MAX}(1, J - K), J \\
\quad \quad A(M + I, J) = \text{matrix}(I, J) \\
10 \quad \text{CONTINUE} \\
20 \quad \text{CONTINUE}
\]

On entry with \(UPLO = 'L'\) or \('l'\), the leading \((K + 1)\) by \(N\) part of the array \(A\) must contain the lower triangular band part of the symmetric matrix, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first subdiagonal starting at position 1 in row 2, and so on. The bottom right \(K\) by \(K\) triangle of the array \(A\) is not referenced. The following program segment transfers the lower triangular part of a symmetric band matrix from conventional full matrix storage to band storage:

\[
\text{DO } 20, J = 1, N \\
\quad M = 1 - J \\
\quad \text{DO } 10, I = J, \text{MIN}(N, J + K) \\
\quad \quad A(M + I, J) = \text{matrix}(I, J) \\
10 \quad \text{CONTINUE} \\
20 \quad \text{CONTINUE}
\]

Unchanged on exit.

\(LDA\) On entry, \(LDA\) specifies the first dimension of \(A\) as declared in the calling (sub) program; \(LDA\) must be at least \((K + 1)\); unchanged on exit.

\(X\) A vector of dimension at least \((1 + (N-1) \times \text{abs(INCX)})\); on entry, the incremented array \(X\) must contain the vector \(x\); unchanged on exit.

\(INCX\) On entry, \(INCX\) specifies the increment for the elements of \(X\); \(INCX\) must not be 0; unchanged on exit.

\(BETA\) On entry, \(BETA\) specifies the scalar \(\beta\); unchanged on exit.

\(Y\) A vector of dimension at least \((1 + (N-1) \times \text{abs(INCY)})\); on entry, the incremented array \(Y\) must contain the vector \(y\); on exit, \(Y\) is overwritten by the updated vector \(y\).

\(INCY\) On entry, \(INCY\) specifies the increment for the elements of \(Y\); \(INCY\) must not be 0; unchanged on exit.

---

**SSPMV or DSPMV Subroutine**

**Purpose**

Performs matrix-vector operations using a packed symmetric matrix.

**Library**

BLAS Library (libblas.a)

**FORTRAN Syntax**

```
SUBROUTINE SSPMV(UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY)

REAL ALPHA, BETA
INTEGER INCX, INCY, N
CHARACTER*1 UPLO
REAL AP(*), X(*)
Y(*)
```

---

732  Technical Reference, Volume 2: Base Operating System and Extensions
SUBROUTINE DSPMV(UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY)
DOUBLE PRECISION ALPHA, BETA
INTEGER INCX, INCY, N
CHARACTER*1 UPL0
DOUBLE PRECISION AP(*), X(*), Y(*)

Description
The SSPMV or DSPMV subroutine performs the matrix-vector operation:

\[ y := \alpha \cdot A \cdot x + \beta \cdot y \]

where alpha and beta are scalars, x and y are \( N \) element vectors and \( A \) is an \( N \) by \( N \) symmetric matrix, supplied in packed form.

Parameters

**UPLO** On entry, **UPLO** specifies whether the upper or lower triangular part of the matrix \( A \) is supplied in the packed array \( AP \) as follows:

- **UPLO** = 'U' or 'u'
  - The upper triangular part of \( A \) is supplied in \( AP \).
- **UPLO** = 'L' or 'l'
  - The lower triangular part of \( A \) is supplied in \( AP \).

Unchanged on exit.

**N** On entry, **N** specifies the order of the matrix \( A \); \( N \) must be at least 0; unchanged on exit.

**ALPHA** On entry, **ALPHA** specifies the scalar alpha; unchanged on exit.

**AP** A vector of dimension at least \((N \cdot (N+1))/2\); on entry with **UPLO** = 'U' or 'u', the array \( AP \) must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that \( AP(1) \) contains \( A(1,1) \), \( AP(2) \) and \( AP(3) \) contain \( A(1,2) \) and \( A(2,2) \) respectively, and so on; on entry with **UPLO** = 'L' or 'l', the array \( AP \) must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that \( AP(1) \) contains \( A(1,1) \), \( AP(2) \) and \( AP(3) \) contain \( A(2,1) \) and \( A(3,1) \) respectively, and so on; unchanged on exit.

**X** A vector of dimension at least \((1 + (N-1) \times \text{abs(INCX)})\); on entry, the incremented array \( X \) must contain the \( N \) element vector \( x \); unchanged on exit.

**INCX** On entry, **INCX** specifies the increment for the elements of \( X \); **INCX** must not be 0; unchanged on exit.

**BETA** On entry, **BETA** specifies the scalar beta; when **BETA** is supplied as 0 then \( Y \) need not be set on input; unchanged on exit.

**Y** A vector of dimension at least \((1 + (N-1) \times \text{abs(INCY)})\); on entry, the incremented array \( Y \) must contain the \( N \) element vector \( y \); on exit, \( Y \) is overwritten by the updated vector \( y \).

**INCY** On entry, **INCY** specifies the increment for the elements of \( Y \); **INCY** must not be 0; unchanged on exit.

---

**STRMV, DTRMV, CTRMV, or ZTRMV Subroutine**

**Purpose**
Performs matrix-vector operations using a triangular matrix.

**Library**
BLAS Library (libblas.a)

**FORTRAN Syntax**

```
SUBROUTINE STRMV(UPLO, TRANS, DIAG, N, A, LDA, X, INCX)
```
The STRMV, DTRMV, CTRMV, or ZTRMV subroutine performs one of the matrix-vector operations:

\[ x := A * x \]

OR

\[ x := A' * x \]

where \( x \) is an \( N \) element vector and \( A \) is an \( N \) by \( N \) unit, or non-unit, upper or lower triangular matrix.

**Parameters**

**UPLO**  
On entry, **UPLO** specifies whether the matrix is an upper or lower triangular matrix as follows:

- **UPLO** = 'U' or 'u'
  - \( A \) is an upper triangular matrix.
- **UPLO** = 'L' or 'l'
  - \( A \) is a lower triangular matrix.

Unchanged on exit.

**TRANS**  
On entry, **TRANS** specifies the operation to be performed as follows:

- **TRANS** = 'N' or 'n'
  - \( x := A * x \)
- **TRANS** = 'T' or 't'
  - \( x := A' * x \)
- **TRANS** = 'C' or 'c'
  - \( x := A' * x \)

Unchanged on exit.

**DIAG**  
On entry, **DIAG** specifies whether or not \( A \) is unit triangular as follows:

- **DIAG** = 'U' or 'u'
  - \( A \) is assumed to be unit triangular.
- **DIAG** = 'N' or 'n'
  - \( A \) is not assumed to be unit triangular.

Unchanged on exit.
On entry, \( N \) specifies the order of the matrix \( A \); \( N \) must be at least 0; unchanged on exit.

An array of dimension \((LDA, N)\); on entry with \( UPLO = 'U' \) or 'u', the leading \( N \) by \( N \) upper triangular part of the array \( A \) must contain the upper triangular matrix and the strictly lower triangular part of \( A \) is not referenced; on entry with \( UPLO = 'L' \) or 'l', the leading \( N \) by \( N \) lower triangular part of the array \( A \) must contain the lower triangular matrix and the strictly upper triangular part of \( A \) is not referenced. When \( DIAG = 'U' \) or 'u', the diagonal elements of \( A \) are not referenced, but are assumed to be unity; unchanged on exit.

On entry, \( LDA \) specifies the first dimension of \( A \) as declared in the calling (sub) program. \( LDA \) must be at least \( \max(1, N) \); unchanged on exit.

A vector of dimension at least \((1 + (N-1) * \text{abs(INCX)})\). On entry, the incremented array \( X \) must contain the \( N \) element vector \( x \); on exit, \( X \) is overwritten with the transformed vector \( x \).

On entry, \( INCX \) specifies the increment for the elements of \( X \); \( INCX \) must not be 0; unchanged on exit.

STBMV, DTBMV, CTBMV, or ZTBMV Subroutine

Purpose
Performs matrix-vector operations using a triangular band matrix.

Library
BLAS Library (libblas.a)

FORTRAN Syntax

```fortran
SUBROUTINE STBMV_UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
INTEGER INCX, K, LDA, N
CHARACTER*1 DIAG, TRANS, UPLO
REAL (LDA, *), X(*)
SUBROUTINE DTBMV_UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
INTEGER INCX, K, LDA, N
CHARACTER*1 DIAG, TRANS, UPLO
DOUBLE PRECISION (LDA, *), X(*)
SUBROUTINE CTBMV_UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
INTEGER INCX, K, LDA, N
CHARACTER*1 DIAG, TRANS, UPLO
COMPLEX A(LDA, *), X(*)
SUBROUTINE ZTBMV_UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
INTEGER INCX, K, LDA, N
CHARACTER*1 DIAG, TRANS, UPLO
COMPLEX*16 A(LDA, *), X(*)
```

Description
The STBMV, DTBMV, CTBMV, or ZTBMV subroutine performs one of the matrix-vector operations:

\[
x := A \cdot x \]

OR

\[
x := A' \cdot x \]

where \( x \) is an \( N \) element vector and \( A \) is an \( N \) by \( N \) unit, or non-unit, upper or lower triangular band matrix, with \((K + 1)\) diagonals.
Parameters

**UPLO**  On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

- **UPLO = 'U' or 'u'**  
  A is an upper triangular matrix.

- **UPLO = 'L' or 'l'**  
  A is a lower triangular matrix.

Unchanged on exit.

**TRANS**  On entry, TRANS specifies the operation to be performed as follows:

- **TRANS = 'N' or 'n'**  
  \( x := A \times x \)

- **TRANS = 'T' or 't'**  
  \( x := A^\top \times x \)

- **TRANS = 'C' or 'c'**  
  \( x := A^\top \times x \)

Unchanged on exit.

**DIAG**  On entry, DIAG specifies whether or not A is unit triangular as follows:

- **DIAG = 'U' or 'u'**  
  A is assumed to be unit triangular.

- **DIAG = 'N' or 'n'**  
  A is not assumed to be unit triangular.

Unchanged on exit.

**N**  On entry, N specifies the order of the matrix A; N must be at least 0; unchanged on exit.

**K**  On entry with UPLO = 'U' or 'u', K specifies the number of superdiagonals of the matrix A; on entry with UPLO = 'L' or 'l', K specifies the number of subdiagonals of the matrix A. K must satisfy 0 ≤ K; unchanged on exit.

**A**  An array of dimension (LDA, N). On entry with UPLO = 'U' or 'u', the leading (K + 1) by N part of the array A must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (K + 1) of the array, the first superdiagonal starting at position 2 in row K, and so on. The top left K by K triangle of the array A is not referenced. The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:

```plaintext
DO 20, J = 1, N
   M = K + 1 - J
   DO 10, I = MAX( 1, J - K ), J
      A( M + I, J ) = matrix( I, J )
   10 CONTINUE
20 CONTINUE
```

On entry with UPLO = 'L' or 'l', the leading (K + 1) by N part of the array A must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first subdiagonal starting at position 1 in row 2, and so on. The bottom right K by K triangle of the array A is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

```plaintext
DO 20, J = 1, N
   M = 1 - J
   DO 10, I = J, MIN( N, J + K )
      A( M + I, J ) = matrix( I, J )
   10 CONTINUE
20 CONTINUE
```

When DIAG = 'U' or 'u' the elements of the array A corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity; unchanged on exit.
**LDA**

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program; LDA must be at least \((K+1)\); unchanged on exit.

**X**

A vector of dimension at least \((1+(N-1) \times \text{abs(INCX)})\); on entry, the incremented array X must contain the \(N\) element vector \(x\); on exit, X is overwritten with the transformed vector \(x\).

**INCX**

On entry, INCX specifies the increment for the elements of \(X\); INCX must not be 0; unchanged on exit.

---

**STPMV, DTPMV, CTPMV, or ZTPMV Subroutine**

**Purpose**

Performs matrix-vector operations on a packed triangular matrix.

**Library**

BLAS Library (libblas.a)

**FORTRAN Syntax**

```
SUBROUTINE STPMV(UPLO, TRANS, DIAG, N, AP, X, INCX)
INTEGER INCX, N
CHARACTER*1 DIAG, TRANS, UPLO
REAL AP(*), X(*)

SUBROUTINE DTPMV(UPLO, TRANS, DIAG, N, AP, X, INCX)
INTEGER INCX, N
CHARACTER*1 DIAG, TRANS, UPLO
DOUBLE PRECISION AP(*), X(*)

SUBROUTINE CTPMV(UPLO, TRANS, DIAG, N, AP, X, INCX)
INTEGER INCX, N
CHARACTER*1 DIAG, TRANS, UPLO
COMPLEX AP(*), X(*)

SUBROUTINE ZTPMV(UPLO, TRANS, DIAG, N, AP, X, INCX)
INTEGER INCX, N
CHARACTER*1 DIAG, TRANS, UPLO
COMPLEX*16 AP(*), X(*)
```

**Description**

The STPMV, DTPMV, CTPMV, or ZTPMV subroutine performs one of the matrix-vector operations:

\[ x := A \times x \]

OR

\[ x := A' \times x \]

where \(x\) is an \(N\) element vector and \(A\) is an \(N\) by \(N\) unit, or non-unit, upper or lower triangular matrix, supplied in packed form.
Parameters

**UPLO**  On entry, **UPLO** specifies whether the matrix is an upper or lower triangular matrix as follows:

**UPLO** = 'U' or 'u'

A is an upper triangular matrix.

**UPLO** = 'L' or 'l'

A is a lower triangular matrix.

Unchanged on exit.

**TRANS**  On entry, **TRANS** specifies the operation to be performed as follows:

**TRANS** = 'N' or 'n'

\[ x := A * x \]

**TRANS** = 'T' or 't'

\[ x := A^T * x \]

**TRANS** = 'C' or 'c'

\[ x := A^H * x \]

Unchanged on exit.

**DIAG**  On entry, **DIAG** specifies whether or not **A** is unit triangular as follows:

**DIAG** = 'U' or 'u'

A is assumed to be unit triangular.

**DIAG** = 'N' or 'n'

A is not assumed to be unit triangular.

Unchanged on exit.

**N**  On entry, **N** specifies the order of the matrix **A**; **N** must be at least 0; unchanged on exit.

**AP**  A vector of dimension at least \( (N \times (N+1))/2 \). On entry with **UPLO** = 'U' or 'u', the array **AP** must contain the upper triangular matrix packed sequentially, column by column, so that **AP**(1) contains **A**(1,1), **AP**(2) and **AP**(3) contain **A**(1,2) and **A**(2,2) respectively, and so on. On entry with **UPLO** = 'L' or 'l', the array **AP** must contain the lower triangular matrix packed sequentially, column by column, so that **AP**(1) contains **A**(1,1), **AP**(2) and **AP**(3) contain **A**(2,1) and **A**(3,1) respectively, and so on. When **DIAG** = 'U' or 'u', the diagonal elements of **A** are not referenced, but are assumed to be unity; unchanged on exit.

**X**  A vector of dimension at least \( (1 + (N-1) \times \text{abs}(\text{INCX})) \); on entry, the incremented array **X** must contain the **N** element vector **x**; on exit, **X** is overwritten with the transformed vector **x**.

**INCX**  On entry, **INCX** specifies the increment for the elements of **X**; **INCX** must not be 0; unchanged on exit.

STRSV, DTRSV, CTRSV, or ZTRSV Subroutine

Purpose
Solves system of equations.

Library
BLAS Library (libblas.a)

FORTRAN Syntax

SUBROUTINE STRSV(UPLO, TRANS, DIAG, 
N, A, LDA, X, INCX)
INTEGER INCX, LDA, N
CHARACTER*1 DIAG, TRANS, UPLO
REAL A(LDA,*), X(*)
SUBROUTINE DTRSV(UPLO, TRANS, DIAG, 
N, A, LDA, X, INCX)
INTEGER INCX, LDA, N
CHARACTER*1 DIAG, TRANS, UPLO
DOUBLE PRECISION A(LDA,*) , X(*)

SUBROUTINE CTRSV(UPLO, TRANS, DIAG, 
N, A, LDA, X, INCX)
INTEGER INCX, LDA, N
CHARACTER*1 DIAG, TRANS, UPLO
COMPLEX A(LDA,*) , X(*)

SUBROUTINE ZTRSV(UPLO, TRANS, DIAG, 
N, A, LDA, X, INCX)
INTEGER INCX, LDA, N
CHARACTER*1 DIAG, TRANS, UPLO
COMPLEX*16 A(LDA,*) , X(*)

Description
The STRSV, DTRSV, CTRSV, or ZTRSV subroutine solves one of the systems of equations:

A * x = b

OR

A' * x = b

where b and x are N element vectors and A is an N by N unit, or non-unit, upper or lower triangular matrix.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

Parameters

UPLO On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u'
A is an upper triangular matrix.

UPLO = 'L' or 'l'
A is a lower triangular matrix.

Unchanged on exit.

TRANS On entry, TRANS specifies the equations to be solved as follows:

TRANS = 'N' or 'n'
A * x = b

TRANS = 'T' or 't'
A' * x = b

TRANS = 'C' or 'c'
A' * x = b

Unchanged on exit.
On entry, \( \text{DIAG} \) specifies whether or not \( A \) is unit triangular as follows:

\( \text{DIAG} = 'U' \) or \( 'u' \)

\( A \) is assumed to be unit triangular.

\( \text{DIAG} = 'N' \) or \( 'n' \)

\( A \) is not assumed to be unit triangular.

Unchanged on exit.

\( N \)

On entry, \( N \) specifies the order of the matrix \( A \); \( N \) must be at least 0; unchanged on exit.

\( A \)

An array of dimension ( \( LDA, N \) ); on entry with \( \text{UPLO} = 'U' \) or \( 'u' \), the leading \( N \) by \( N \) upper triangular part of the array \( A \) must contain the upper triangular matrix and the strictly lower triangular part of \( A \) is not referenced. On entry with \( \text{UPLO} = 'L' \) or \( 'l' \), the leading \( N \) by \( N \) lower triangular part of the array \( A \) must contain the lower triangular matrix and the strictly upper triangular part of \( A \) is not referenced. When \( \text{DIAG} = 'U' \) or \( 'u' \), the diagonal elements of \( A \) are not referenced, but are assumed to be unity; unchanged on exit.

\( LDA \)

On entry, \( LDA \) specifies the first dimension of \( A \) as declared in the calling (sub) program; \( LDA \) must be at least \( \max(1, N) \); unchanged on exit.

\( X \)

A vector of dimension at least \( (1 + (N-1) \times \text{abs(INCX)} ) \); on entry, the incremented array \( X \) must contain the \( N \) element right-hand side vector \( b \); on exit, \( X \) is overwritten with the solution vector \( x \).

\( INCX \)

On entry, \( INCX \) specifies the increment for the elements of \( X \); \( INCX \) must not be 0; unchanged on exit.

**STBSV, DTBSV, CTBSV, or ZTBSV Subroutine**

**Purpose**

Solves system of equations.

**Library**

BLAS Library (libblas.a)

**FORTRAN Syntax**

SUBROUTINE STBSV(UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
INTEGER INCX, K, LDA, N
CHARACTER*1 DIAG, TRANS, UPLO
REAL A(LDA,*), X(*)

SUBROUTINE DTBSV(UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
INTEGER INCX, K, LDA, N
CHARACTER*1 DIAG, TRANS, UPLO
DOUBLE PRECISION A(LDA,*), X(*)

SUBROUTINE CTBSV(UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
INTEGER INCX, K, LDA, N
CHARACTER*1 DIAG, TRANS, UPLO
COMPLEX A(LDA,*), X(*)

SUBROUTINE ZTBSV(UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)
INTEGER INCX, K, LDA, N
CHARACTER*1 DIAG, TRANS, UPLO
COMPLEX*16 A(LDA,*), X(*)

**Description**

The STBSV, DTBSV, CTBSV, or ZTBSV subroutine solves one of the systems of equations:

\[ A \times x = b \]
\[ A' \times x = b \]

where \( b \) and \( x \) are \( N \) element vectors and \( A \) is an \( N \) by \( N \) unit, or non-unit, upper or lower triangular band matrix, with \( (K + 1) \) diagonals.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

**Parameters**

**UPLO** On entry, \( UPLO \) specifies whether the matrix is an upper or lower triangular matrix as follows:

- \( UPLO = \text{'U'} \) or \text{'u'}
  - \( A \) is an upper triangular matrix.
- \( UPLO = \text{'L'} \) or \text{'l'}
  - \( A \) is a lower triangular matrix.

Unchanged on exit.

**TRANS** On entry, \( TRANS \) specifies the equations to be solved as follows:

- \( TRANS = \text{'N'} \) or \text{'n'}
  - \( A \times x = b \)
- \( TRANS = \text{'T'} \) or \text{'t'}
  - \( A' \times x = b \)
- \( TRANS = \text{'C'} \) or \text{'c'}
  - \( A' \times x = b \)

Unchanged on exit.

**DIAG** On entry, \( DIAG \) specifies whether \( A \) is unit triangular as follows:

- \( DIAG = \text{'U'} \) or \text{'u'}
  - \( A \) is assumed to be unit triangular.
- \( DIAG = \text{'N'} \) or \text{'n'}
  - \( A \) is not assumed to be unit triangular.

Unchanged on exit.

**N** On entry, \( N \) specifies the order of the matrix \( A \); \( N \) must be at least 0; unchanged on exit.

**K** On entry with \( UPLO = \text{'U'} \) or \text{'u'}, \( K \) specifies the number of superdiagonals of the matrix \( A \). On entry with \( UPLO = \text{'L'} \) or \text{'l'}, \( K \) specifies the number of subdiagonals of the matrix \( A \); \( K \) must satisfy \( 0 \leq K \); unchanged on exit.
A

An array of dimension \((LDA, N)\). On entry with \(UPLO = 'U'\) or \('u'\), the leading \((K + 1)\) by \(N\) part of the array \(A\) must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row \((K + 1)\) of the array, the first superdiagonal starting at position 2 in row \(K\), and so on. The top left \(K\) by \(K\) triangle of the array \(A\) is not referenced.

The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:

```fortran
DO 20, J = 1, N
   M = K + 1 - J
   DO 10, I = MAX( 1, J - K ), J
      A( M + I, J ) = matrix( I, J )
   10 CONTINUE
20 CONTINUE
```

On entry with \(UPLO = 'L'\) or \('l'\), the leading \((K + 1)\) by \(N\) part of the array \(A\) must contain the lower triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row 1 of the array, the first subdiagonal starting at position 1 in row 2, and so on. The bottom right \(K\) by \(K\) triangle of the array \(A\) is not referenced.

The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

```fortran
DO 20, J = 1, N
   M = 1 - J
   DO 10, I = J, MIN( N, J + K )
      A( M + I, J ) = matrix( I, J )
   10 CONTINUE
20 CONTINUE
```

When \(DIAG = 'U'\) or \('u'\) the elements of the array \(A\) corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity. Unchanged on exit.

\(LDA\)

On entry, \(LDA\) specifies the first dimension of \(A\) as declared in the calling (sub) program; \(LDA\) must be at least \((K + 1)\); unchanged on exit.

\(X\)

A vector of dimension at least \((1 + (N-1) \times \text{abs(INCX)})\); on entry, the incremented array \(X\) must contain the \(N\) element right-hand side vector \(b\); on exit, \(X\) is overwritten with the solution vector \(x\).

\(INCX\)

On entry, \(INCX\) specifies the increment for the elements of \(X\); \(INCX\) must not be 0; unchanged on exit.

---

**STPSV, DTPSV, CTPSV, or ZTPSV Subroutine**

**Purpose**

Solves systems of equations.

**Library**

BLAS Library (libblas.a)

**FORTRAN Syntax**

```fortran
SUBROUTINE STPSV(UPLO, TRANS, DIAG, N, AP, X, INCX)
   INTEGER INCX, N
   CHARACTER*1 DIAG, TRANS, UPLO
   REAL AP(*), X(*)
SUBROUTINE DTPSV(UPLO, TRANS, DIAG, N, AP, X, INCX)
   INTEGER INCX, N
   CHARACTER*1 DIAG, TRANS, UPLO
   DOUBLE PRECISION AP(*), X(*)
```
SUBROUTINE CTPSV(UPLO, TRANS, DIAG, N, AP, X, INCX)
INTEGER INCX, N
CHARACTER*1 DIAG, TRANS, UPLO
COMPLEX AP(*), X(*)

SUBROUTINE ZTPSV(UPLO, TRANS, DIAG, N, AP, X, INCX)
INTEGER INCX, N
CHARACTER*1 DIAG, TRANS, UPLO
COMPLEX*16 AP(*), X(*)

Description
The STPSV, DTPSV, DTTPS, or ZTPSV subroutine solves one of the systems of equations:

\[ A \cdot x = b \]

OR

\[ A' \cdot x = b \]

where \( b \) and \( x \) are \( N \) element vectors and \( A \) is an \( N \times N \) unit, or non-unit, upper or lower triangular matrix, supplied in packed form.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

Parameters

UPLO On entry, UPLO specifies whether the matrix is an upper or lower triangular matrix as follows:

- \( UPLO = 'U' \) or \('u' \)
  - \( A \) is an upper triangular matrix.
- \( UPLO = 'L' \) or \('l' \)
  - \( A \) is a lower triangular matrix.

Unchanged on exit.

TRANS On entry, TRANS specifies the equations to be solved as follows:

- \( TRANS = 'N' \) or \('n' \)
  - \( A \cdot x = b \)
- \( TRANS = 'T' \) or \('t' \)
  - \( A' \cdot x = b \)
- \( TRANS = 'C' \) or \('c' \)
  - \( A' \cdot x = b \)

Unchanged on exit.

DIAG On entry, DIAG specifies whether or not \( A \) is unit triangular as follows:

- \( DIAG = 'U' \) or \('u' \)
  - \( A \) is assumed to be unit triangular.
- \( DIAG = 'N' \) or \('n' \)
  - \( A \) is not assumed to be unit triangular.

Unchanged on exit.

N On entry, \( N \) specifies the order of the matrix \( A \); \( N \) must be at least 0; unchanged on exit.
A vector of dimension at least \(( N \times (N+1) / 2 )\); on entry with \textit{UPLO} = 'U' or 'u', the array \textit{AP} must contain the upper triangular matrix packed sequentially, column by column, so that \textit{AP}(1) contains \(A(1,1)\), \textit{AP}(2) and \textit{AP}(3) contain \(A(1,2)\) and \(A(2,2)\) respectively, and so on. Before entry with \textit{UPLO} = 'L' or 'l', the array \textit{AP} must contain the lower triangular matrix packed sequentially, column by column, so that \textit{AP}(1) contains \(A(1,1)\), \textit{AP}(2) and \textit{AP}(3) contain \(A(2,1)\) and \(A(3,1)\) respectively, and so on. When \textit{DIAG} = 'U' or 'u', the diagonal elements of \(A\) are not referenced, but are assumed to be unity; unchanged on exit.

\textbf{SGER or DGER Subroutine}

\textbf{Purpose}

Performs the rank 1 operation.

\textbf{Library}

BLAS Library (libblas.a)

\textbf{FORTRAN Syntax}

\begin{verbatim}
SUBROUTINE SGER(M, N, ALPHA, X, INCX, Y, INCY, A, LDA)
  REAL ALPHA, A(LDA,*)
  INTEGER INX, INCY, LDA, M, N

SUBROUTINE DGER(M, N, ALPHA, X, INCX, Y, INCY, A, LDA)
  DOUBLE PRECISION ALPHA, A(LDA,*)
  INTEGER INX, INCY, LDA, M, N
\end{verbatim}

\textbf{Description}

The \textit{SGER} or \textit{DGER} subroutine performs the rank 1 operation:

\[ A := \alpha \times x \times y' + A \]

where \(\alpha\) is a scalar, \(x\) is an \(M\) element vector, \(y\) is an \(N\) element vector and \(A\) is an \(M\) by \(N\) matrix.

\textbf{Parameters}

\begin{itemize}
  \item \(M\) On entry, \(M\) specifies the number of rows of the matrix \(A\); \(M\) must be at least 0; unchanged on exit.
  \item \(N\) On entry, \(N\) specifies the number of columns of the matrix \(A\); \(N\) must be at least 0; unchanged on exit.
  \item \(\text{ALPHA}\) On entry, \(\text{ALPHA}\) specifies the scalar \(\alpha\); unchanged on exit.
  \item \(X\) A vector of dimension at least \((1 + (M-1) \times \text{abs(INCX)})\); on entry, the incremented array \(X\) must contain the \(M\) element vector \(x\); unchanged on exit.
  \item \(\text{INCX}\) On entry, \(\text{INCX}\) specifies the increment for the elements of \(X\); \(\text{INCX}\) must not be 0; unchanged on exit.
  \item \(Y\) A vector of dimension at least \((1 + (N-1) \times \text{abs(INCY)})\); on entry, the incremented array \(Y\) must contain the \(N\) element vector \(y\); unchanged on exit.
  \item \(\text{INCY}\) On entry, \(\text{INCY}\) specifies the increment for the elements of \(Y\); \(\text{INCY}\) must not be 0; unchanged on exit.
  \item \(A\) An array of dimension \((LDA, N)\); on entry, the leading \(M\) by \(N\) part of the array \(A\) must contain the matrix of coefficients; on exit, \(A\) is overwritten by the updated matrix.
  \item \(LDA\) On entry, \(LDA\) specifies the first dimension of \(A\) as declared in the calling (sub) program; \(LDA\) must be at least \(\max(1, M)\); unchanged on exit.
\end{itemize}
CGERU or ZGERU Subroutine

Purpose
Performs the rank 1 operation.

Library
BLAS Library (libblas.a)

FORTRAN Syntax
SUBROUTINE CGERU(M, N, ALPHA, X, INCX, Y, INCY, A, LDA)
  COMPLEX ALPHA
  INTEGER INCX, INCY, LDA, M, N
  COMPLEX A(LDA,*), X(*), Y(*)
END SUBROUTINE

Description
The CGERU subroutine performs the rank 1 operation:

\[ A := \alpha x y' + A \]

where \( \alpha \) is a scalar, \( x \) is an \( M \) element vector, \( y \) is an \( N \) element vector and \( A \) is an \( M \) by \( N \) matrix.

Parameters
- \( M \): On entry, \( M \) specifies the number of rows of the matrix \( A \); \( M \) must be at least 0; unchanged on exit.
- \( N \): On entry, \( N \) specifies the number of columns of the matrix \( A \); \( N \) must be at least 0; unchanged on exit.
- \( \alpha \): On entry, \( \alpha \) specifies the scalar alpha; unchanged on exit.
- \( x \): A vector of dimension at least \( 1 + (M - 1) \times \text{abs(INCX)} \); on entry, the incremented array \( x \) must contain the \( M \) element vector \( x \); unchanged on exit.
- \( \text{INCX} \): On entry, \( \text{INCX} \) specifies the increment for the elements of \( x \); \( \text{INCX} \) must not be 0; unchanged on exit.
- \( y \): A vector of dimension at least \( 1 + (N - 1) \times \text{abs(INCY)} \); on entry, the incremented array \( y \) must contain the \( N \) element vector \( y \); unchanged on exit.
- \( \text{INCY} \): On entry, \( \text{INCY} \) specifies the increment for the elements of \( y \); \( \text{INCY} \) must not be 0; unchanged on exit.
- \( A \): An array of dimension \( (LDA, N) \); on entry, the leading \( M \) by \( N \) part of the array \( A \) must contain the matrix of coefficients; on exit, \( A \) is overwritten by the updated matrix.
- \( LDA \): On entry, \( LDA \) specifies the first dimension of \( A \) as declared in the calling (sub) program; \( LDA \) must be at least \( \max(1, M) \); unchanged on exit.

CGERC or ZGERC Subroutine

Purpose
Performs the rank 1 operation.

Library
BLAS Library (libblas.a)
FORTRAN Syntax

SUBROUTINE CGERC( M, N, ALPHA, X, INCX, Y, INCY, A, LDA)
COMPLEX ALPHA
INTEGER INCX, INCY, LDA, M, N
COMPLEX A(LDA,*), X(*), Y(*)

SUBROUTINE ZGERC
COMPLEX*16 ALPHA
INTEGER INCX, INCY, LDA, M, N
COMPLEX*16 A(LDA,*), X(*), Y(*)

Description
The CGERC or ZGERC subroutine performs the rank 1 operation:

\[ A := \alpha \times x \times \text{conjg}(y^\top) + A \]

where \( \alpha \) is a scalar, \( x \) is an \( M \) element vector, \( y \) is an \( N \) element vector and \( A \) is an \( M \) by \( N \) matrix.

Parameters

\( M \)  
On entry, \( M \) specifies the number of rows of the matrix \( A \); \( M \) must be at least 0; unchanged on exit.

\( N \)  
On entry, \( N \) specifies the number of columns of the matrix \( A \); \( N \) must be at least 0; unchanged on exit.

\( \alpha \)  
On entry, \( \alpha \) specifies the scalar \( \alpha \); unchanged on exit.

\( X \)  
A vector of dimension at least \( (1 + (M-1) \times \text{abs(INCX)}) \); on entry, the incremented array \( X \) must contain the \( M \) element vector \( x \); unchanged on exit.

\( \text{INCX} \)  
On entry, \( \text{INCX} \) specifies the increment for the elements of \( X \); \( \text{INCX} \) must not be 0; unchanged on exit.

\( Y \)  
A vector of dimension at least \( (1 + (N-1) \times \text{abs(INCY)}) \); on entry, the incremented array \( Y \) must contain the \( N \) element vector \( y \); unchanged on exit.

\( \text{INCY} \)  
On entry, \( \text{INCY} \) specifies the increment for the elements of \( Y \); \( \text{INCY} \) must not be 0; unchanged on exit.

\( A \)  
An array of dimension \( (LDA, N) \); on entry, the leading \( M \) by \( N \) part of the array \( A \) must contain the matrix of coefficients; on exit, \( A \) is overwritten by the updated matrix.

\( LDA \)  
On entry, \( LDA \) specifies the first dimension of \( A \) as declared in the calling (sub) program; \( LDA \) must be at least \( \max(1, M) \); unchanged on exit.

CHER or ZHER Subroutine

Purpose
Performs the Hermitian rank 1 operation.

Library
BLAS Library (libblas.a)

FORTRAN Syntax

SUBROUTINE CHER( UPLO, N, ALPHA, X, INCX, A, LDA)
REAL ALPHA
INTEGER INCX, LDA, N
CHARACTER*1 UPLO
COMPLEX A(LDA,*), X(*)

SUBROUTINE ZHER
REAL*16 ALPHA
INTEGER INCX, LDA, N
CHARACTER*1 UPLO
COMPLEX*16 A(LDA,*), X(*)

746 Technical Reference, Volume 2: Base Operating System and Extensions
SUBROUTINE ZHER(UPLO, N, ALPHA, X, INCX, A, LDA)
DOUBLE PRECISION ALPHA
INTEGER INCX, LDA, N
CHARACTER*1 UPLO
COMPLEX*16 A(LDA,*), X(*)

Description
The CHER or ZHER subroutine performs the Hermitian rank 1 operation:

\[ A := \alpha \times x \times \text{conjg}(x^\top) + A \]

where alpha is a real scalar, x is an \( N \) element vector and \( A \) is an \( N \) by \( N \) Hermitian matrix.

Parameters

UPLO On entry, UPLO specifies whether the upper or lower triangular part of the array \( A \) is to be referenced as follows:

- \( \text{UPLO} = \text{'U'} \) or \( \text{'u'} \)
  - Only the upper triangular part of \( A \) is to be referenced.
- \( \text{UPLO} = \text{'L'} \) or \( \text{'l'} \)
  - Only the lower triangular part of \( A \) is to be referenced.

Unchanged on exit.

N On entry, \( N \) specifies the order of the matrix \( A \); \( N \) must be at least 0; unchanged on exit.

ALPHA On entry, ALPHA specifies the scalar alpha; unchanged on exit.

X A vector of dimension at least (1 + (N-1) \times \text{abs(INCX)}); on entry, the incremented array \( X \) must contain the \( N \) element vector \( x \); unchanged on exit.

INCX On entry, INCX specifies the increment for the elements of \( X \); INCX must not be 0; unchanged on exit.

A An array of dimension (LDA, N); on entry with \( \text{UPLO} = \text{'U'} \) or \( \text{'u'} \), the leading \( N \) by \( N \) upper triangular part of the array \( A \) must contain the upper triangular part of the Hermitian matrix and the strictly lower triangular part of \( A \) is not referenced. On exit, the upper triangular part of the array \( A \) is overwritten by the upper triangular part of the updated matrix. On entry with \( \text{UPLO} = \text{'L'} \) or \( \text{'l'} \), the leading \( N \) by \( N \) lower triangular part of the array \( A \) must contain the lower triangular part of the Hermitian matrix and the strictly upper triangular part of \( A \) is not referenced. On exit, the lower triangular part of the array \( A \) is overwritten by the lower triangular part of the updated matrix. The imaginary parts of the diagonal elements need not be set, they are assumed to be 0, and on exit they are set to 0.

LDA On entry, LDA specifies the first dimension of \( A \) as declared in the calling (sub) program; LDA must be at least max(1, N); unchanged on exit.

CHPR or ZHPR Subroutine

Purpose
Performs the Hermitian rank 1 operation.

Library
BLAS Library (libblas.a)

FORTRAN Syntax
SUBROUTINE CHPR(UPLO, N, ALPHA, X, INCX, AP)
REAL ALPHA
SUBROUTINE ZHPR(UPLO, N, ALPHA, X, INCX, AP)
DOUBLE PRECISION ALPHA
INTEGER INCX, N
CHARACTER*1 UPLO
COMPLEX*16 AP(*), X(*)

Description
The CHPR or ZHPR subroutine performs the Hermitian rank 1 operation:

\[ A := \alpha \cdot x \cdot \text{conjg}(x') + A \]

where \( \alpha \) is a real scalar, \( x \) is an \( N \) element vector and \( A \) is an \( N \) by \( N \) Hermitian matrix, supplied in packed form.

Parameters

UPLO On entry, UPLO specifies whether the upper or lower triangular part of the matrix \( A \) is supplied in the packed array \( AP \) as follows:
- \( UPLO = 'U' \) or 'u'
The upper triangular part of \( A \) is supplied in \( AP \).
- \( UPLO = 'L' \) or 'l'
The lower triangular part of \( A \) is supplied in \( AP \).

Unchanged on exit.

N On entry, \( N \) specifies the order of the matrix \( A \); \( N \) must be at least 0; unchanged on exit.

ALPHA On entry, ALPHA specifies the scalar \( \alpha \); unchanged on exit.

X A vector of dimension at least \((1 + (N-1) \cdot \text{abs(INCX)})\); on entry, the incremented array \( X \) must contain the \( N \) element vector \( x \); unchanged on exit.

INCY On entry, INCX specifies the increment for the elements of \( X \); INCX must not be 0; unchanged on exit.

AP A vector of dimension at least \((N \cdot (N+1) / 2)\); on entry with \( UPLO = 'U' \) or 'u', the array \( AP \) must contain the upper triangular part of the Hermitian matrix packed sequentially, column by column, so that \( AP(1) \) contains \( A(1,1) \), \( AP(2) \) and \( AP(3) \) contain \( A(1,2) \) and \( A(2,2) \) respectively, and so on. On exit, the array \( AP \) is overwritten by the upper triangular part of the updated matrix. On entry with \( UPLO = 'L' \) or 'l', the array \( AP \) must contain the lower triangular part of the Hermitian matrix packed sequentially, column by column, so that \( AP(1) \) contains \( A(1,1) \), \( AP(2) \) and \( AP(3) \) contain \( A(2,1) \) and \( A(3,1) \) respectively, and so on. On exit, the array \( AP \) is overwritten by the lower triangular part of the updated matrix. The imaginary parts of the diagonal elements need not be set, they are assumed to be 0, and on exit they are set to 0.

CHER2 or ZHER2 Subroutine

Purpose
Performs the Hermitian rank 2 operation.

Library
BLAS Library (libblas.a)

FORTRAN Syntax
SUBROUTINE CHER2(UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA)
The ChER2 or ZHER2 subroutine performs the Hermitian rank 2 operation:

\[ A := \alpha x^* \, \text{conj}(y^*) + \text{conj}(\alpha) y^* \, \text{conj}(x^*) + A \]

where \( \alpha \) is a scalar, \( x \) and \( y \) are \( N \) element vectors and \( A \) is an \( N \times N \) Hermitian matrix.

### Parameters

- **UPLO**
  - On entry, **UPLO** specifies whether the upper or lower triangular part of the array \( A \) is to be referenced as follows:
  - **UPLO** = 'U' or 'u'
    - Only the upper triangular part of \( A \) is to be referenced.
  - **UPLO** = 'L' or 'l'
    - Only the lower triangular part of \( A \) is to be referenced.

- **N**
  - On entry, \( N \) specifies the order of the matrix \( A \); \( N \) must be at least 0; unchanged on exit.

- **ALPHA**
  - On entry, **ALPHA** specifies the scalar alpha; unchanged on exit.

- **X**
  - A vector of dimension at least \((1 + (N-1) \times \text{abs(INCX)})\); on entry, the incremented vector \( X \) must contain the \( N \) element vector \( x \); unchanged on exit.

- **INCX**
  - On entry, **INCX** specifies the increment for the elements of \( X \); **INCX** must not be 0; unchanged on exit.

- **Y**
  - A vector of dimension at least \((1 + (N-1) \times \text{abs(INCY)})\); on entry, the incremented vector \( Y \) must contain the \( N \) element vector \( y \); unchanged on exit.

- **INCY**
  - On entry, **INCY** specifies the increment for the elements of \( Y \); **INCY** must not be 0; unchanged on exit.

- **A**
  - An array of dimension \((LDA, N)\); on entry with **UPLO** = 'U' or 'u', the leading \( N \) by \( N \) upper triangular part of the array \( A \) must contain the upper triangular part of the Hermitian matrix and the strictly lower triangular part of \( A \) is not referenced. On exit, the upper triangular part of the array \( A \) is overwritten by the upper triangular part of the updated matrix. On entry with **UPLO** = 'L' or 'l', the leading \( N \) by \( N \) lower triangular part of the array \( A \) must contain the lower triangular part of the Hermitian matrix and the strictly upper triangular part of \( A \) is not referenced. On exit, the lower triangular part of the array \( A \) is overwritten by the lower triangular part of the updated matrix. The imaginary parts of the diagonal elements need not be set; they are assumed to be 0, and on exit they are set to 0.

- **LDA**
  - On entry, **LDA** specifies the first dimension of \( A \) as declared in the calling (sub) program; **LDA** must be at least \( \max(1, N) \); unchanged on exit.

### CHPR2 or ZHPR2 Subroutine

**Purpose**
- Performs the Hermitian rank 2 operation.

**Library**
- BLAS Library (libblas.a)
FORTRAN Syntax

SUBROUTINE CHPR2 (UPLO, N, ALPHA, X, INCX, Y, INCY, AP)
  COMPLEX ALPHA
  INTEGER INCX, INCY, N
  CHARACTER*1 UPLO
  COMPLEX AP(*), X(*), Y(*)

SUBROUTINE ZHPR2
  COMPLEX*16 ALPHA
  INTEGER INCX, INCY, N
  CHARACTER*1 UPLO
  COMPLEX*16 AP(*), X(*), Y(*)

Description

The CHPR2 or ZHPR2 subroutine performs the Hermitian rank 2 operation:

\[ A := \alpha \cdot x \cdot \text{conjg}(y') + \text{conjg}(\alpha) \cdot y \cdot \text{conjg}(x') + A \]

where \( \alpha \) is a scalar, \( x \) and \( y \) are \( N \) element vectors and \( A \) is an \( N \) by \( N \) Hermitian matrix, supplied in packed form.

Parameters

- **UPLO** On entry, \( \text{UPLO} \) specifies whether the upper or lower triangular part of the matrix \( A \) is supplied in the packed array \( AP \) as follows:
  - \( \text{UPLO} = 'U' \) or \( 'u' \): The upper triangular part of \( A \) is supplied in \( AP \).
  - \( \text{UPLO} = 'L' \) or \( 'l' \): The lower triangular part of \( A \) is supplied in \( AP \).

- **N** On entry, \( N \) specifies the order of the matrix \( A \); \( N \) must be at least 0; unchanged on exit.

- **ALPHA** On entry, \( \text{ALPHA} \) specifies the scalar \( \alpha \); unchanged on exit.

- **X** A vector of dimension at least \( 1 + (N-1) \cdot \text{abs(INCX)} \); on entry, the incremented array \( X \) must contain the \( N \) element vector \( x \); unchanged on exit.

- **INCX** On entry, \( \text{INCX} \) specifies the increment for the elements of \( X \); \( \text{INCX} \) must not be 0; unchanged on exit.

- **Y** A vector of dimension at least \( 1 + (N-1) \cdot \text{abs(INCY)} \); on entry, the incremented array \( Y \) must contain the \( N \) element vector \( y \); unchanged on exit.

- **INCY** On entry, \( \text{INCY} \) specifies the increment for the elements of \( Y \); \( \text{INCY} \) must not be 0; unchanged on exit.

- **AP** A vector of dimension at least \( (N \cdot (N+1))/2 \); on entry with \( \text{UPLO} = 'U' \) or \( 'u' \), the array \( AP \) must contain the upper triangular part of the Hermitian matrix packed sequentially, column by column, so that \( AP(1) \) contains \( A(1,1) \), \( AP(2) \) and \( AP(3) \) contain \( A(1,2) \) and \( A(2,2) \) respectively, and so on. On exit, the array \( AP \) is overwritten by the upper triangular part of the updated matrix. On entry with \( \text{UPLO} = 'L' \) or \( 'l' \), the array \( AP \) must contain the lower triangular part of the Hermitian matrix packed sequentially, column by column, so that \( AP(1) \) contains \( A(1,1) \), \( AP(2) \) and \( AP(3) \) contain \( A(2,1) \) and \( A(3,1) \) respectively, and so on. On exit, the array \( AP \) is overwritten by the lower triangular part of the updated matrix. The imaginary parts of the diagonal elements need not be set; they are assumed to be 0, and on exit they are set to 0.

SSYR or DSYR Subroutine

Purpose

Performs the symmetric rank 1 operation.
Library
BLAS Library (libblas.a)

FORTRAN Syntax

SUBROUTINE SSYR(UPLO, N, ALPHA, X, INCX, A, LDA)
REAL ALPHA
INTEGER INCX, LDA, N
CHARACTER*1 UPLO
REAL A(LDA,*), X(*)

SUBROUTINE DSYR(UPLO, N, ALPHA, X, INCX, A, LDA)
DOUBLE PRECISION ALPHA
INTEGER INCX, LDA, N
CHARACTER*1 UPLO
DOUBLE PRECISION A(LDA,*), X(*)

Description
The SSYR or DSYR subroutine performs the symmetric rank 1 operation:

\[ A := \alpha \cdot x \cdot x' + A \]

where alpha is a real scalar, x is an \( N \) element vector and \( A \) is an \( N \times N \) symmetric matrix.

Parameters

UPLO On entry, \( \text{UPLO} \) specifies whether the upper or lower triangular part of the array \( A \) is to be referenced as follows:

\( \text{UPLO} = 'U' \) or \'u'  
Only the upper triangular part of \( A \) is to be referenced.

\( \text{UPLO} = 'L' \) or \'l'  
Only the lower triangular part of \( A \) is to be referenced.

Unchanged on exit.

N On entry, \( N \) specifies the order of the matrix \( A \); \( N \) must be at least 0; unchanged on exit.

ALPHA On entry, \( \text{ALPHA} \) specifies the scalar alpha; unchanged on exit.

X A vector of dimension at least \( 1 + (N-1) \cdot \text{abs(INCX)} \); on entry, the incremented array \( X \) must contain the \( N \) element vector \( x \); unchanged on exit.

INCX On entry, \( \text{INCX} \) specifies the increment for the elements of \( X \); \( \text{INCX} \) must not be 0; unchanged on exit.

A An array of dimension \( (LDA, N) \); on entry with \( \text{UPLO} = 'U' \) or \'u' \), the leading \( N \) by \( N \) upper triangular part of the array \( A \) must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of \( A \) is not referenced. On exit, the upper triangular part of the array \( A \) is overwritten by the upper triangular part of the updated matrix. On entry with \( \text{UPLO} = 'L' \) or \'l' \), the leading \( N \) by \( N \) lower triangular part of the array \( A \) must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of \( A \) is not referenced. On exit, the lower triangular part of the array \( A \) is overwritten by the lower triangular part of the updated matrix.

LDA On entry, \( \text{LDA} \) specifies the first dimension of \( A \) as declared in the calling (sub) program; \( \text{LDA} \) must be at least \( \max(1, N) \); unchanged on exit.

SSPR or DSPR Subroutine

Purpose
Performs the symmetric rank 1 operation.
Library
BLAS Library (libblas.a)

FORTRAN Syntax

SUBROUTINE SSPR(UPLO, N, ALPHA, X, INCX, AP)
  REAL ALPHA
  INTEGER INCX, N
  CHARACTER*1 UPLO
  REAL AP(*), X(*)

SUBROUTINE DSPR(UPLO, N, ALPHA, X, INCX, AP)
  DOUBLE PRECISION ALPHA
  INTEGER INCX, N
  CHARACTER*1 UPLO
  DOUBLE PRECISION AP(*), X(*)

Description
The SSPR or DSPR subroutine performs the symmetric rank 1 operation:

\[ A := \alpha \cdot x \cdot x' + A \]

where \( \alpha \) is a real scalar, \( x \) is an \( N \) element vector and \( A \) is an \( N \) by \( N \) symmetric matrix, supplied in packed form.

Parameters

**UPLO**
On entry, **UPLO** specifies whether the upper or lower triangular part of the matrix \( A \) is supplied in the packed array \( AP \) as follows:

- **UPLO** = 'U' or 'u'
  - The upper triangular part of \( A \) is supplied in \( AP \).
- **UPLO** = 'L' or 'l'
  - The lower triangular part of \( A \) is supplied in \( AP \).

Unchanged on exit.

**N**
On entry, \( N \) specifies the order of the matrix \( A \); \( N \) must be at least 0; unchanged on exit.

**ALPHA**
On entry, **ALPHA** specifies the scalar \( \alpha \); unchanged on exit.

**X**
A vector of dimension at least \( (1 + (N-1) \times \text{abs(INCX)}) \); on entry, the incremented array \( X \) must contain the \( N \) element vector \( x \); unchanged on exit.

**INCX**
On entry, **INCX** specifies the increment for the elements of \( X \); **INCX** must not be 0; unchanged on exit.

**AP**
A vector of dimension at least \( (N \times (N+1)) / 2 \); on entry with **UPLO** = 'U' or 'u', the array **AP** must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that **AP** contains \( A(1,1) \), **AP**(2) and **AP**(3) contain \( A(1,2) \) and \( A(2,2) \) respectively, and so on. On exit, the array **AP** is overwritten by the upper triangular part of the updated matrix. On entry with **UPLO** = 'L' or 'l', the array **AP** must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that **AP** contains \( A(1,1) \), **AP**(2) and **AP**(3) contain \( A(2,1) \) and \( A(3,1) \) respectively, and so on. On exit, the array **AP** is overwritten by the lower triangular part of the updated matrix.

SSYR2 or DSYR2 Subroutine

Purpose
Performs the symmetric rank 2 operation.
Library

BLAS Library (libblas.a)

FORTRAN Syntax

```fortran
SUBROUTINE SSYR2(UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA)
REAL ALPHA
INTEGER INCX, INCY, LDA, N
CHARACTER*1 UPLO
REAL A(LDA,*), X(*), Y(*)
SUBROUTINE DSYR2(UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA)
DOUBLE PRECISION ALPHA
INTEGER INCX, INCY, LDA, N
CHARACTER*1 UPLO
DOUBLE PRECISION A(LDA,*), X(*), Y(*)
```

Description

The SSYR2 or DSYR2 subroutine performs the symmetric rank 2 operation:

\[
A := \alpha * x * y' + \alpha * y * x' + A
\]

where alpha is a scalar, x and y are \( N \) element vectors and \( A \) is an \( N \) by \( N \) symmetric matrix.

Parameters

- **UPLO** On entry, \( UPLO \) specifies whether the upper or lower triangular part of the array \( A \) is to be referenced as follows:
  - \( UPLO = \text{'U'} \) or \( \text{'u'} \)
    - Only the upper triangular part of \( A \) is to be referenced.
  - \( UPLO = \text{'L'} \) or \( \text{'l'} \)
    - Only the lower triangular part of \( A \) is to be referenced.
  Unchanged on exit.

- **N** On entry, \( N \) specifies the order of the matrix \( A \); \( N \) must be at least 0; unchanged on exit.

- **ALPHA** On entry, \( ALPHA \) specifies the scalar alpha; unchanged on exit.

- **X** A vector of dimension at least \( (1 + (N-1) \times \text{abs(INCX)}) \); on entry, the incremented array \( X \) must contain the \( N \) element vector \( x \); unchanged on exit.

- **INCX** On entry, \( INCX \) specifies the increment for the elements of \( X \); \( INCX \) must not be 0; unchanged on exit.

- **Y** A vector of dimension at least \( (1 + (N-1) \times \text{abs(INCY)}) \); on entry, the incremented array \( Y \) must contain the \( N \) element vector \( y \); unchanged on exit.

- **INCY** On entry, \( INCY \) specifies the increment for the elements of \( Y \); \( INCY \) must not be 0; unchanged on exit.

- **A** An array of dimension \( (LDA, N) \); on entry with \( UPLO = \text{'U'} \) or \( \text{'u'} \), the leading \( N \) by \( N \) upper triangular part of the array \( A \) must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of \( A \) is not referenced. On exit, the upper triangular part of the array \( A \) is overwritten by the upper triangular part of the updated matrix. On entry with \( UPLO = \text{'L'} \) or \( \text{'l'} \), the leading \( N \) by \( N \) lower triangular part of the array \( A \) must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of \( A \) is not referenced. On exit, the lower triangular part of the array \( A \) is overwritten by the lower triangular part of the updated matrix.

- **LDA** On entry, \( LDA \) specifies the first dimension of \( A \) as declared in the calling (sub) program; \( LDA \) must be at least \( \max(1, N) \); unchanged on exit.
SSPR2 or DSPR2 Subroutine

Purpose
Performs the symmetric rank 2 operation.

Library
BLAS Library (libblas.a)

FORTRAN Syntax
SUBROUTINE SSPR2(UPLO, N, ALPHA, X, INCX, Y, INCY, AP)
REAL ALPHA
INTEGER INCX, INCY, N
CHARACTER*1 UPLO
REAL AP(*), X(*), Y(*)
SUBROUTINE DSPR2(UPLO, N, ALPHA, X, INCX, Y, INCY, AP)
DOUBLE PRECISION ALPHA
INTEGER INCX, INCY, N
CHARACTER*1 UPLO
DOUBLE PRECISION AP(*), X(*), Y(*)

Description
The SSPR2 or DSPR2 subroutine performs the symmetric rank 2 operation:

A := alpha * x * y’ + alpha * y * x’ + A

where alpha is a scalar, x and y are N element vectors and A is an N by N symmetric matrix, supplied in packed form.

Parameters

UPLO On entry, UPLO specifies whether the upper or lower triangular part of the matrix A is supplied in the packed array AP as follows:

UPLO = 'U' or 'u'
  The upper triangular part of A is supplied in AP.

UPLO = 'L' or 'l'
  The lower triangular part of A is supplied in AP.

Unchanged on exit.

N On entry, N specifies the order of the matrix A; N must be at least 0; unchanged on exit.

ALPHA On entry, ALPHA specifies the scalar alpha; unchanged on exit.

X A vector of dimension at least (1 + (N-1) * abs(INCX) ); on entry, the incremented array X must contain the N element vector x; unchanged on exit.

INCY On entry, INCY specifies the increment for the elements of X; INCY must not be 0; unchanged on exit.

Y A vector of dimension at least (1 + (N-1) * abs(INCY) ); on entry, the incremented array Y must contain the N element vector y; unchanged on exit.

INCY On entry, INCY specifies the increment for the elements of Y; INCY must not be 0; unchanged on exit.
A vector of dimension at least \((N \times (N+1))/2\); on entry with \(UPLO = 'U'\) or \(u'\), the array \(AP\) must contain the upper triangular part of the symmetric matrix packed sequentially, column by column, so that \(AP(1)\) contains \(A(1,1)\), \(AP(2)\) and \(AP(3)\) contain \(A(1,2)\) and \(A(2,2)\) respectively, and so on. On exit, the array \(AP\) is overwritten by the upper triangular part of the updated matrix. On entry with \(UPLO = 'L'\) or \(l'\), the array \(AP\) must contain the lower triangular part of the symmetric matrix packed sequentially, column by column, so that \(AP(1)\) contains \(A(1,1)\), \(AP(2)\) and \(AP(3)\) contain \(A(2,1)\) and \(A(3,1)\) respectively, and so on. On exit, the array \(AP\) is overwritten by the lower triangular part of the updated matrix.

**SGEMM, DGEMM, CGEMM, or ZGEMM Subroutine**

**Purpose**
Performs matrix-matrix operations on general matrices.

**Library**
BLAS Library (libblas.a)

**FORTRAN Syntax**
```
SUBROUTINE SGEMM(TRANSA, TRANSB, M, N, K,
    ALPHA, A, LDA, B, LDB, BETA, C, LDC)
CHARACTER*1 TRANSA, TRANSB
INTEGER M, N, K, LDA, LDB, LDC
REAL ALPHA, BETA
REAL A(LDA, *), B(LDB, *), C(LDC, *)
```
```
SUBROUTINE DGEMM(TRANSA, TRANSB, M, N, K,
    ALPHA, A, LDA, B, LDB, BETA, C, LDC)
CHARACTER*1 TRANSA, TRANSB
INTEGER M, N, K, LDA, LDB, LDC
DOUBLE PRECISION ALPHA, BETA
DOUBLE PRECISION A(LDA, *), B(LDB, *), C(LDC, *)
```
```
SUBROUTINE CGEMM(TRANSA, TRANSB, M, N, K,
    ALPHA, A, LDA, B, LDB, BETA, C, LDC)
CHARACTER*1 TRANSA, TRANSB
INTEGER M, N, K, LDA, LDB, LDC
COMPLEX ALPHA, BETA
COMPLEX A(LDA, *), B(LDB, *), C(LDC, *)
```
```
SUBROUTINE ZGEMM(TRANSA, TRANSB, M, N, K,
    ALPHA, A, LDA, B, LDB, BETA, C, LDC)
CHARACTER*1 TRANSA, TRANSB
INTEGER M, N, K, LDA, LDB, LDC
COMPLEX*16 ALPHA, BETA
COMPLEX*16 A(LDA, *), B(LDB, *), C(LDC, *)
```

**Description**
The **SGEMM, DGEMM, CGEMM, or ZGEMM** subroutine performs one of the matrix-matrix operations:

\[
C := \alpha \times \text{op}(A) \times \text{op}(B) + \beta \times C
\]

where \(\text{op}(X)\) is one of \(\text{op}(X) = X\) or \(\text{op}(X) = X'\), \(\alpha\) and \(\beta\) are scalars, and \(A, B\) and \(C\) are matrices, with \(\text{op}(A)\) an \(M\) by \(K\) matrix, \(\text{op}(B)\) a \(K\) by \(N\) matrix and \(C\) an \(M\) by \(N\) matrix.
Parameters

\texttt{TRANSA} On entry, \texttt{TRANSA} specifies the form of \text{op}( A ) to be used in the matrix multiplication as follows:

\texttt{TRANSA} = 'N' or 'n'
\hspace{1cm} \text{op}( A ) = A
\texttt{TRANSA} = 'T' or 't'
\hspace{1cm} \text{op}( A ) = A'
\texttt{TRANSA} = 'C' or 'c'
\hspace{1cm} \text{op}( A ) = A'

Unchanged on exit.

\texttt{TRANSB} On entry, \texttt{TRANSB} specifies the form of \text{op}( B ) to be used in the matrix multiplication as follows:

\texttt{TRANSB} = 'N' or 'n'
\hspace{1cm} \text{op}( B ) = B
\texttt{TRANSB} = 'T' or 't'
\hspace{1cm} \text{op}( B ) = B'
\texttt{TRANSB} = 'C' or 'c'
\hspace{1cm} \text{op}( B ) = B'

Unchanged on exit.

\texttt{M} On entry, \texttt{M} specifies the number of rows of the matrix \text{op}( A ) and of the matrix \texttt{C}; \texttt{M} must be at least 0; unchanged on exit.

\texttt{N} On entry, \texttt{N} specifies the number of columns of the matrix \text{op}( B ) and the number of columns of the matrix \texttt{C}; \texttt{N} must be at least 0; unchanged on exit.

\texttt{K} On entry, \texttt{K} specifies the number of columns of the matrix \text{op}( A ) and the number of rows of the matrix \text{op}( B ); \texttt{K} must be at least 0; unchanged on exit.

\texttt{ALPHA} On entry, \texttt{ALPHA} specifies the scalar alpha; unchanged on exit.

\texttt{A} An array of dimension ( \texttt{LDA}, \texttt{KA} ), where \texttt{KA} is \texttt{K} when \texttt{TRANSA} = 'N' or 'n', and is \texttt{M} otherwise; on entry with \texttt{TRANSA} = 'N' or 'n', the leading \texttt{M} by \texttt{K} part of the array \texttt{A} must contain the matrix \texttt{A}, otherwise the leading \texttt{K} by \texttt{M} part of the array \texttt{A} must contain the matrix \texttt{A}; unchanged on exit.

\texttt{LDA} On entry, \texttt{LDA} specifies the first dimension of \texttt{A} as declared in the calling (sub) program. When \texttt{TRANSA} = 'N' or 'n' then \texttt{LDA} must be at least \text{max}( 1, \texttt{M} ), otherwise \texttt{LDA} must be at least \text{max}( 1, \texttt{K} ); unchanged on exit.

\texttt{B} An array of dimension ( \texttt{LDB}, \texttt{KB} ) where \texttt{KB} is \texttt{N} when \texttt{TRANSB} = 'N' or 'n', and is \texttt{K} otherwise; on entry with \texttt{TRANSB} = 'N' or 'n', the leading \texttt{K} by \texttt{N} part of the array \texttt{B} must contain the matrix \texttt{B}, otherwise the leading \texttt{N} by \texttt{K} part of the array \texttt{B} must contain the matrix \texttt{B}; unchanged on exit.

\texttt{LDB} On entry, \texttt{LDB} specifies the first dimension of \texttt{B} as declared in the calling (sub) program. When \texttt{TRANSB} = 'N' or 'n' then \texttt{LDB} must be at least \text{max}( 1, \texttt{K} ), otherwise \texttt{LDB} must be at least \text{max}( 1, \texttt{N} ); unchanged on exit.

\texttt{BETA} On entry, \texttt{BETA} specifies the scalar beta. When \texttt{BETA} is supplied as 0 then \texttt{C} need not be set on input; unchanged on exit.

\texttt{C} An array of dimension ( \texttt{LDC}, \texttt{N} ); on entry, the leading \texttt{M} by \texttt{N} part of the array \texttt{C} must contain the matrix \texttt{C}, except when \texttt{beta} is 0, in which case \texttt{C} need not be set on entry; on exit, the array \texttt{C} is overwritten by the \texttt{M} by \texttt{N} matrix ( \text{alpha} * \text{op}( \texttt{A} ) * \text{op}( \texttt{B} ) + \text{beta} * \texttt{C} ).

\texttt{LDC} On entry, \texttt{LDC} specifies the first dimension of \texttt{C} as declared in the calling (sub) program; \texttt{LDC} must be at least \text{max}( 1, \texttt{M} ); unchanged on exit.

---

**SSYMM, DSYMM, CSYMM, or ZSYMM Subroutine**

**Purpose**

Performs matrix-matrix matrix operations on symmetric matrices.
Library
BLAS Library (libblas.a)

FORTRAN Syntax

SUBROUTINE SSYMM(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
CHARACTER*1 SIDE, UPLO
INTEGER M, N, LDA, LDB, LDC
REAL ALPHA, BETA
REAL A(LDA,*), B(LDB,*), C(LDC,*)

SUBROUTINE DSYMM(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
CHARACTER*1 SIDE, UPLO
INTEGER M, N, LDA, LDB, LDC
DOUBLE PRECISION ALPHA, BETA
DOUBLE PRECISION A(LDA,*), B(LDB,*), C(LDC,*)

SUBROUTINE CSYMM(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
CHARACTER*1 SIDE, UPLO
INTEGER M, N, LDA, LDB, LDC
COMPLEX ALPHA, BETA
COMPLEX A(LDA,*), B(LDB,*), C(LDC,*)

SUBROUTINE ZSYMM(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
CHARACTER*1 SIDE, UPLO
INTEGER M, N, LDA, LDB, LDC
COMPLEX*16 ALPHA, BETA
COMPLEX*16 A(LDA,*), B(LDB,*), C(LDC,*)

Description

The SSYMM, DSYMM, CSYMM, or ZSYMM subroutine performs one of the matrix-matrix operations:

\[ C := \alpha \cdot A \cdot B + \beta \cdot C \]

OR

\[ C := \alpha \cdot B \cdot A + \beta \cdot C \]

where alpha and beta are scalars, A is a symmetric matrix and B and C are M by N matrices.

Parameters

SIDE On entry, SIDE specifies whether the symmetric matrix A appears on the left or right in the operation as follows:

SIDE = 'L' or 'l'
\[ C := \alpha \cdot A \cdot B + \beta \cdot C \]

SIDE = 'R' or 'r'
\[ C := \alpha \cdot B \cdot A + \beta \cdot C \]

Unchanged on exit.
On entry, \textit{UPLO} specifies whether the upper or lower triangular part of the symmetric matrix \( A \) is to be referenced as follows:

\textit{UPLO} = \textit{‘U’} or \textit{‘u’}
Only the upper triangular part of the symmetric matrix is to be referenced.

\textit{UPLO} = \textit{‘L’} or \textit{‘l’}
Only the lower triangular part of the symmetric matrix is to be referenced.

Unchanged on exit.

\textit{M}
On entry, \( M \) specifies the number of rows of the matrix \( C \); \( M \) must be at least 0; unchanged on exit.

\textit{N}
On entry, \( N \) specifies the number of columns of the matrix \( C \); \( N \) must be at least 0; unchanged on exit.

\textit{ALPHA}
On entry, \( \textit{ALPHA} \) specifies the scalar \( \alpha \); unchanged on exit.

\textit{A}
An array of dimension ( \( \textit{LDA}, \textit{KA} \) ), where \( \textit{KA} \) is \( M \) when \textit{SIDE} = \textit{‘L’} or \textit{‘l’} and is \( N \) otherwise; on entry with \textit{SIDE} = \textit{‘L’} or \textit{‘l’}, the \( M \) by \( M \) part of the array \( A \) must contain the symmetric matrix, such that when \textit{UPLO} = \textit{‘U’} or \textit{‘u’}, the leading \( M \) by \( M \) upper triangular part of the array \( A \) must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of \( A \) is not referenced, and when \textit{UPLO} = \textit{‘L’} or \textit{‘l’}, the leading \( M \) by \( M \) lower triangular part of the array \( A \) must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of \( A \) is not referenced. On entry with \textit{SIDE} = \textit{‘R’} or \textit{‘r’}, the \( N \) by \( N \) part of the array \( A \) must contain the symmetric matrix, such that when \textit{UPLO} = \textit{‘U’} or \textit{‘u’}, the leading \( N \) by \( N \) upper triangular part of the array \( A \) must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of \( A \) is not referenced, and when \textit{UPLO} = \textit{‘L’} or \textit{‘l’}, the leading \( N \) by \( N \) lower triangular part of the array \( A \) must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of \( A \) is not referenced; unchanged on exit.

\textit{LDA}
On entry, \( \textit{LDA} \) specifies the first dimension of \( A \) as declared in the calling (sub) program. When \textit{SIDE} = \textit{‘L’} or \textit{‘l’} then \( \textit{LDA} \) must be at least max( 1, \( M \) ), otherwise \( \textit{LDA} \) must be at least max( 1, \( N \) ); unchanged on exit.

\textit{B}
An array of dimension ( \( \textit{LDB}, N \) ); on entry, the leading \( M \) by \( N \) part of the array \( B \) must contain the matrix \( B \); unchanged on exit.

\textit{LDB}
On entry, \( \textit{LDB} \) specifies the first dimension of \( B \) as declared in the calling (sub) program; \( \textit{LDB} \) must be at least max( 1, \( M \) ); unchanged on exit.

\textit{BETA}
On entry, \( \textit{BETA} \) specifies the scalar \( \beta \); when \( \textit{BETA} \) is supplied as 0 then \( \textit{C} \) need not be set on input; unchanged on exit.

\textit{C}
An array of dimension ( \( \textit{LDC}, N \) ); on entry, the leading \( M \) by \( N \) part of the array \( C \) must contain the matrix \( C \), except when \( \textit{BETA} \) is 0, in which case \( \textit{C} \) need not be set on entry; on exit, the array \( \textit{C} \) is overwritten by the \( M \) by \( N \) updated matrix.

\textit{LDC}
On entry, \( \textit{LDC} \) specifies the first dimension of \( C \) as declared in the calling (sub) program; \( \textit{LDC} \) must be at least max( 1, \( M \) ); unchanged on exit.

---

**CHEMM or ZHEMM Subroutine**

**Purpose**
Performs matrix-matrix operations on Hermitian matrices.

**Library**
BLAS Library (libblas.a)

**FORTRAN Syntax**

```
SUBROUTINE CHEMM(SIDE, UPLO, M, N, ALPHA, A, 
LDA, B, LDB, BETA, C, LDC)
CHARACTER*1 SIDE, UPLO
INTEGER M, N, LDA, LDB, LDC
COMPLEX ALPHA, BETA
COMPLEX A(LDA,*), B(LDB,*), C(LDC,*)
```
SUBROUTINE ZHEMM(SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
CHARACTER*1 SIDE,UPLO
INTEGER M,N,LDA,LDB,LDC
COMPLEX*16 ALPHA,BETA
COMPLEX*16 A(LDA,*), B(LDB,*), C(LDC,*)

Purpose
The CHEMM or ZHEMM subroutine performs one of the matrix-matrix operations:

\[ C := \alpha \cdot A \cdot B + \beta \cdot C \]  OR  \[ C := \alpha \cdot B \cdot A + \beta \cdot C \]

where alpha and beta are scalars, A is an Hermitian matrix, and B and C are M by N matrices.

Parameters

SIDE  On entry, SIDE specifies whether the Hermitian matrix A appears on the left or right in the operation as follows:

SIDE = 'L' or 'l'
\[ C := \alpha \cdot A \cdot B + \beta \cdot C \]
SIDE = 'R' or 'r'
\[ C := \alpha \cdot B \cdot A + \beta \cdot C \]

Unchanged on exit.

UPLO  On entry, UPLO specifies whether the upper or lower triangular part of the Hermitian matrix A is to be referenced as follows:

UPLO = 'U' or 'u'
Only the upper triangular part of the Hermitian matrix is to be referenced.

UPLO = 'L' or 'l'
Only the lower triangular part of the Hermitian matrix is to be referenced.

Unchanged on exit.

M  On entry, M specifies the number of rows of the matrix C; M must be at least 0; unchanged on exit.

N  On entry, N specifies the number of columns of the matrix C; N must be at least 0; unchanged on exit.

ALPHA  On entry, ALPH A specifies the scalar alpha; unchanged on exit.

A  An array of dimension ( LDA, KA ), where KA is M when SIDE = 'L' or 'l' and is N otherwise; on entry with SIDE = 'L' or 'l', the M by M part of the array A must contain the Hermitian matrix, such that when UPLO = 'U' or 'u', the leading M by M upper triangular part of the array A must contain the upper triangular part of the Hermitian matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading M by M lower triangular part of the array A must contain the lower triangular part of the Hermitian matrix and the strictly upper triangular part of A is not referenced; on entry with SIDE = 'R' or 'r', the N by N part of the array A must contain the Hermitian matrix, such that when UPLO = 'U' or 'u', the leading N by N upper triangular part of the array A must contain the upper triangular part of the Hermitian matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading N by N lower triangular part of the array A must contain the lower triangular part of the Hermitian matrix and the strictly upper triangular part of A is not referenced. The imaginary parts of the diagonal elements need not be set, they are assumed to be 0; unchanged on exit.

LDA  On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least max( 1, M ), otherwise LDA must be at least max( 1, N ); unchanged on exit.

B  An array of dimension ( LDB, N ); on entry, the leading M by N part of the array B must contain the matrix B; unchanged on exit.

LDB  On entry, LDB specifies the first dimension of B as declared in the calling (sub) program; LDB must be at least max( 1, M ); unchanged on exit.
On entry, $BETA$ specifies the scalar beta. When $BETA$ is supplied as 0 then $C$ need not be set on input; unchanged on exit.

An array of dimension ($LDC, N$); on entry, the leading $M$ by $N$ part of the array $C$ must contain the matrix $C$, except when beta is 0, in which case $C$ need not be set on entry; on exit, the array $C$ is overwritten by the $M$ by $N$ updated matrix.

On entry, $LDC$ specifies the first dimension of $C$ as declared in the calling (sub) program; $LDC$ must be at least max($1, M$); unchanged on exit.

---

**SSYRK, DSYRK, CSYRK, or ZSYRK Subroutine**

**Purpose**

Perform symmetric rank k operations.

**Library**

BLAS Library (libblas.a)

**FORTRAN Syntax**

```fortran
SUBROUTINE SSYRK(UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)
CHARACTER*1 UPLO, TRANS
INTEGER M, K, LDA, LDC
REAL ALPHA, BETA
REAL A(LDA,*), C(LDC,*)
SUBROUTINE DSYRK(UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)
CHARACTER*1 UPLO, TRANS
INTEGER N, K, LDA, LDC
DOUBLE PRECISION ALPHA, BETA
DOUBLE PRECISION A(LDA,*), C(LDC,*)
SUBROUTINE CSYRK(UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)
CHARACTER*1 UPLO, TRANS
INTEGER N, K, LDA, LDC
COMPLEX ALPHA, BETA
COMPLEX A(LDA,*), C(LDC,*)
SUBROUTINE ZSYRK(UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)
CHARACTER*1 UPLO, TRANS
INTEGER N, K, LDA, LDC
COMPLEX*16 ALPHA, BETA
COMPLEX*16 A(LDA,*), C(LDC,*)
```

**Description**

The **SSYRK**, **DSYRK**, **CSYRK** or **ZSYRK** subroutine performs one of the symmetric rank k operations:

\[
C := \alpha \cdot A \cdot A' + \beta \cdot C
\]

OR

\[
C := \alpha \cdot A' \cdot A + \beta \cdot C
\]

where alpha and beta are scalars, $C$ is an $N$ by $N$ symmetric matrix, and $A$ is an $N$ by $K$ matrix in the first case and a $K$ by $N$ matrix in the second case.
Parameters

UPLO  On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

UPLO = 'U' or 'u'
Only the upper triangular part of C is to be referenced.

UPLO = 'L' or 'l'
Only the lower triangular part of C is to be referenced.

Unchanged on exit.

TRANS  On entry, TRANS specifies the operation to be performed as follows:

TRANS = 'N' or 'n'
C := alpha * A * A' + beta * C

TRANS = 'T' or 't'
C := alpha * A' * A + beta * C

TRANS = 'C' or 'c'
C := alpha * A' * A + beta * C

Unchanged on exit.

N  On entry, N specifies the order of the matrix C; N must be at least 0; unchanged on exit.

K  On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrix A, and on entry with TRANS = 'T' or 't' or 'C' or 'c', K specifies the number of rows of the matrix A; K must be at least 0; unchanged on exit.

ALPHA  On entry, ALPHA specifies the scalar alpha; unchanged on exit.

A  An array of dimension (LDA, KA), where KA is K when TRANS = 'N' or 'n', and is N otherwise; on entry with TRANS = 'N' or 'n', the leading N by K part of the array A must contain the matrix A, otherwise the leading K by N part of the array A must contain the matrix A; unchanged on exit.

LDA  On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n', LDA must be at least max(1, N); otherwise LDA must be at least max(1, K); unchanged on exit.

BETA  On entry, BETA specifies the scalar beta; unchanged on exit.

C  An array of dimension (LDC, N); on entry with UPLO = 'U' or 'u', the leading N by N upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced; on exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix; on entry with UPLO = 'L' or 'l', the leading N by N lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced; on exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.

LDC  On entry, LDC specifies the first dimension of C as declared in the calling (sub) program; LDC must be at least max(1, N); unchanged on exit.

CHERK or ZHERK Subroutine

Purpose
Performs Hermitian rank k operations.

Library
BLAS Library (libblas.a)

FORTRAN Syntax
SUBROUTINE CHERK(UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)
CHARACTER*1 UPLO, TRANS
INTEGER N, K, LDA, LDC
REAL ALPHA, BETA
COMPLEX A(LDA, *), C(LDC, *)

SUBROUTINE ZHERK(UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)
CHARACTER*1 UPLO, TRANS
INTEGER N, K, LDA, LDC
DOUBLE PRECISION ALPHA, BETA
COMPLEX*16 A(LDA, *), C(LDC, *)

Description
The Cherk or ZHERK subroutine performs one of the Hermitian rank k operations:

\[ C := \alpha \cdot A \cdot \text{conjg}(A') + \beta \cdot C \]

OR

\[ C := \alpha \cdot \text{conjg}(A') \cdot A + \beta \cdot C \]

where \( \alpha \) and \( \beta \) are real scalars, \( C \) is an \( N \) by \( N \) Hermitian matrix, and \( A \) is an \( N \) by \( K \) matrix in the first case and a \( K \) by \( N \) matrix in the second case.

Parameters

**UPLO** On entry, **UPLO** specifies whether the upper or lower triangular part of the array \( C \) is to be referenced as follows:

- **UPLO** = 'U' or 'u'
  - Only the upper triangular part of \( C \) is to be referenced.
- **UPLO** = 'L' or 'l'
  - Only the lower triangular part of \( C \) is to be referenced.

Unchanged on exit.

**TRANS** On entry, **TRANS** specifies the operation to be performed as follows:

- **TRANS** = 'N' or 'n'
  - \( C := \alpha \cdot A \cdot \text{conjg}(A') + \beta \cdot C \)
- **TRANS** = 'C' or 'c'
  - \( C := \alpha \cdot \text{conjg}(A') \cdot A + \beta \cdot C \)

Unchanged on exit.

**N** On entry, \( N \) specifies the order of the matrix \( C \); \( N \) must be at least 0; unchanged on exit.

**K** On entry with **TRANS** = 'N' or 'n', \( K \) specifies the number of columns of the matrix \( A \), and on entry with **TRANS** = 'C' or 'c', \( K \) specifies the number of rows of the matrix \( A \); \( K \) must be at least 0; unchanged on exit.

**ALPHA** On entry, **ALPHA** specifies the scalar \( \alpha \); unchanged on exit.

**A** An array of dimension ( \( LDA, KA \) ), where \( KA \) is \( K \) when **TRANS** = 'N' or 'n', and is \( N \) otherwise; on entry with **TRANS** = 'N' or 'n', the leading \( N \) by \( K \) part of the array \( A \) must contain the matrix \( A \), otherwise the leading \( K \) by \( N \) part of the array \( A \) must contain the matrix \( A \); unchanged on exit.

**LDA** On entry, **LDA** specifies the first dimension of \( A \) as declared in the calling (sub) program. When **TRANS** = 'N' or 'n', \( LDA \) must be at least max( 1, \( N \) ); otherwise \( LDA \) must be at least max( 1, \( K \) ); unchanged on exit.

**BETA** On entry, **BETA** specifies the scalar \( \beta \); unchanged on exit.
An array of dimension \((LDC, N)\); on entry with \(UPLO = 'U'\) or \('u'\), the leading \(N\) by \(N\) upper triangular part of the array \(C\) must contain the upper triangular part of the Hermitian matrix and the strictly lower triangular part of \(C\) is not referenced; on exit, the upper triangular part of the array \(C\) is overwritten by the upper triangular part of the updated matrix; on entry with \(UPLO = 'L'\) or \('l'\), the leading \(N\) by \(N\) lower triangular part of the array \(C\) must contain the lower triangular part of the Hermitian matrix and the strictly upper triangular part of \(C\) is not referenced; on exit, the lower triangular part of the array \(C\) is overwritten by the lower triangular part of the updated matrix. The imaginary parts of the diagonal elements need not be set, they are assumed to be 0, and on exit they are set to 0.

**SSYR2K, DSYR2K, CSYR2K, or ZSYR2K Subroutine**

**Purpose**

Performs symmetric rank 2k operations.

**Library**

BLAS Library (libblas.a)

**FORTRAN Syntax**

```
SUBROUTINE SSYR2K(UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
CHARACTER*1 UPLO, TRANS
INTEGER N, K, LDA, LDB, LDC
REAL ALPHA, BETA
REAL A(LDA,*), B(LDB,*), C(LDC,*)
```

```
SUBROUTINE DSYR2K(UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
CHARACTER*1 UPLO, TRANS
INTEGER N, K, LDA, LDB, LDC
DOUBLE PRECISION ALPHA, BETA
DOUBLE PRECISION A(LDA,*), B(LDB,*), C(LDC,*)
```

```
SUBROUTINE CSYR2K(UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
CHARACTER*1 UPLO, TRANS
INTEGER N, K, LDA, LDB, LDC
COMPLEX ALPHA, BETA
COMPLEX A(LDA,*), B(LDB,*), C(LDC,*)
```

```
SUBROUTINE ZSYR2K(UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
CHARACTER*1 UPLO, TRANS
INTEGER N, K, LDA, LDB, LDC
COMPLEX*16 ALPHA, BETA
COMPLEX*16 A(LDA,*), B(LDB,*), C(LDC,*)
```

**Description**

The \texttt{SSYR2K}, \texttt{DSYR2K}, \texttt{CSYR2K}, or \texttt{ZSYR2K} subroutine performs one of the symmetric rank 2k operations:

\[
C := \alpha \ast A \ast B' + \alpha \ast B \ast A' + \beta \ast C
\]

**OR**

\[
C := \alpha \ast A' \ast B + \alpha \ast B' \ast A + \beta \ast C
\]
where alpha and beta are scalars, C is an N by N symmetric matrix, and A and B are N by K matrices in the first case and K by N matrices in the second case.

Parameters

**UPLO**
On entry, **UPLO** specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

- **UPLO** = 'U' or 'u'
  Only the upper triangular part of C is to be referenced.
- **UPLO** = 'L' or 'l'
  Only the lower triangular part of C is to be referenced.

Unchanged on exit.

**TRANS**
On entry, **TRANS** specifies the operation to be performed as follows:

- **TRANS** = 'N' or 'n'
  \[
  C := \alpha \cdot A \cdot B + \alpha \cdot B \cdot A' + \beta \cdot C
  \]
- **TRANS** = 'T' or 't'
  \[
  C := \alpha \cdot A' \cdot B + \alpha \cdot B' \cdot A + \beta \cdot C
  \]

Unchanged on exit.

**N**
On entry, **N** specifies the order of the matrix C; **N** must be at least 0; unchanged on exit.

**K**
On entry with **TRANS** = 'N' or 'n', **K** specifies the number of columns of the matrices A and B, and on entry with **TRANS** = 'T' or 't', **K** specifies the number of rows of the matrices A and B; **K** must be at least 0; unchanged on exit.

**ALPHA**
On entry, **ALPHA** specifies the scalar alpha; unchanged on exit.

**A**
An array of dimension (LDA, KA), where KA is K when **TRANS** = 'N' or 'n', and is N otherwise; on entry with **TRANS** = 'N' or 'n', the leading N by K part of the array A must contain the matrix A, otherwise the leading K by N part of the array A must contain the matrix A; unchanged on exit.

**LDA**
On entry, **LDA** specifies the first dimension of A as declared in the calling (sub) program. When **TRANS** = 'N' or 'n', **LDA** must be at least max(1, N); otherwise **LDA** must be at least max(1, K); unchanged on exit.

**B**
An array of dimension (LDB, KB), where KB is K when **TRANS** = 'N' or 'n', and is N otherwise; on entry with **TRANS** = 'N' or 'n', the leading N by K part of the array B must contain the matrix B, otherwise the leading K by N part of the array B must contain the matrix B; unchanged on exit.

**LDB**
On entry, **LDB** specifies the first dimension of B as declared in the calling (sub) program. When **TRANS** = 'N' or 'n', **LDB** must be at least max(1, N); otherwise **LDB** must be at least max(1, K); unchanged on exit.

**BETA**
On entry, **BETA** specifies the scalar beta; unchanged on exit.

**C**
An array of dimension (LDC, N); on entry with **UPLO** = 'U' or 'u', the leading N by N upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced; on exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix. On entry with **UPLO** = 'L' or 'l', the leading N by N lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced; on exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.

**LDC**
On entry, **LDC** specifies the first dimension of C as declared in the calling (sub) program; **LDC** must be at least max(1, N); unchanged on exit.

**CHER2K or ZHER2K Subroutine**

**Purpose**
Performs Hermitian rank 2k operations.
Library
BLAS Library (libblas.a)

FORTRAN Syntax

SUBROUTINE CHER2K(UPLO, TRANS, N, K, ALPHA,
A, LDA, B, LDB, C, LDC)
CHARACTER*1 UPLO
INTEGER N, K, LDA, LDB, LDC
REAL BETA
COMPLEX ALPHA
COMPLEX A(LDA,*), B(LDB,*), C(LDC,*)
SUBROUTINE ZHER2K(UPLO, TRANS, N, K, ALPHA,
A, LDA, B, LDB, C, LDC)
CHARACTER*1 UPLO
INTEGER N, K, LDA, LDB, LDC
DOUBLE PRECISION BETA
COMPLEX*16 ALPHA
COMPLEX*16 A(LDA,*), B(LDB,*), C(LDC,*)

Description
The CHER2K or ZHER2K subroutine performs one of the Hermitian rank 2k operations:

\[ C := \alpha * A * \text{conjg}(B') + \text{conjg}(\alpha) * B * \text{conjg}(A') + \beta * C \]

OR

\[ C := \alpha * \text{conjg}(A') * B + \text{conjg}(\alpha) * \text{conjg}(B') * A + \beta * C \]

where alpha and beta are scalars with beta real, C is an N by N Hermitian matrix, and A and B are N by K matrices in the first case and K by N matrices in the second case.

Parameters

UPLO On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

- UPLO = 'U' or 'u'
  Only the upper triangular part of C is to be referenced.

- UPLO = 'L' or 'l'
  Only the lower triangular part of C is to be referenced.

Unchanged on exit.

TRANS On entry, TRANS specifies the operation to be performed as follows:

- TRANS = 'N' or 'n'
  \[ C := \alpha * A * \text{conjg}(B') + \text{conjg}(\alpha) * B * \text{conjg}(A') + \beta * C \]

- TRANS = 'C' or 'c'
  \[ C := \alpha * \text{conjg}(A') * B + \text{conjg}(\alpha) * \text{conjg}(B') * A + \beta * C \]

Unchanged on exit.

N On entry, N specifies the order of the matrix C; N must be at least 0; unchanged on exit.

K On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrices A and B, and on entry with TRANS = 'C' or 'c', K specifies the number of rows of the matrices A and B; K must be at least 0; unchanged on exit.

ALPHA On entry, ALPHA specifies the scalar alpha; unchanged on exit.
A
An array of dimension ( LDA, KA ), where KA is K when TRANS = 'N' or 'n', and is N otherwise; on entry with TRANS = 'N' or 'n', the leading N by K part of the array A must contain the matrix A, otherwise the leading K by N part of the array A must contain the matrix A; unchanged on exit.

LDA
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n', LDA must be at least max( 1, N ); otherwise LDA must be at least max( 1, K ); unchanged on exit.

B
An array of dimension ( LDB, KB ), where KB is K when TRANS = 'N' or 'n', and is N otherwise; on entry with TRANS = 'N' or 'n', the leading N by K part of the array B must contain the matrix B, otherwise the leading K by N part of the array B must contain the matrix B; unchanged on exit.

LDB
On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANS = 'N' or 'n', LDB must be at least max( 1, N ); otherwise LDB must be at least max( 1, K ); unchanged on exit.

BETA
On entry, BETA specifies the scalar beta; unchanged on exit.

C
An array of dimension ( LDC, N ); on entry with UPLO = 'U' or 'u', the leading N by N upper triangular part of the array C must contain the upper triangular part of the Hermitian matrix and the strictly lower triangular part of C is not reference; on exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix; on entry with UPLO = 'L' or 'l', the leading N by N lower triangular part of the array C must contain the lower triangular part of the Hermitian matrix and the strictly upper triangular part of C is not referenced; on exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix. The imaginary parts of the diagonal elements need not be set, they are assumed to be 0, and on exit they are set to 0.

LDC
On entry, LDC specifies the first dimension of C as declared in the calling (sub) program; LDC must be at least max( 1, N ); unchanged on exit.

---

**STRMM, DTRMM, CTRMM, or ZTRMM Subroutine**

**Purpose**
Performs matrix-matrix operations on triangular matrices.

**Library**
BLAS Library (libblas.a)

**FORTRAN Syntax**

```fortran
SUBROUTINE STRMM(SIDE, UPLO, TRANSA, DIAG,
M, N, ALPHA, A, LDA, B, LDB)
 CHARACTER*1 SIDE, UPLO, TRANSA, DIAG
 INTEGER M, N, LDA, LDB
 REAL ALPHA
 REAL A(LDA,*), B(LDB,*)

SUBROUTINE DTRMM(SIDE, UPLO, TRANSA, DIAG,
M, N, ALPHA, A, LDA, B, LDB)
 CHARACTER*1 SIDE,UPLO,TRANSA,DIAG
 INTEGER M,N,LDA,LDB
 DOUBLE PRECISION ALPHA
 DOUBLE PRECISION A(LDA,*), B(LDB,*)

SUBROUTINE CTRMM(SIDE, UPLO, TRANSA, DIAG,
M, N, ALPHA, A, LDA, B, LDB)
 CHARACTER*1 SIDE,UPLO,TRANSA,DIAG
 INTEGER M,N,LDA,LDB
 COMPLEX ALPHA
 COMPLEX A(LDA,*), B(LDB,*)

SUBROUTINE ZTRMM(SIDE, UPLO, TRANSA, DIAG,
M, N, ALPHA, A, LDA, B, LDB)
 CHARACTER*1
```

766  Technical Reference, Volume 2: Base Operating System and Extensions
Description

The \texttt{STRMM}, \texttt{DTRMM}, \texttt{CTRMM}, or \texttt{ZTRMM} subroutine performs one of the matrix-matrix operations:

\[ B := \alpha \cdot \text{op}(A) \cdot B \]

OR

\[ B := \alpha \cdot B \cdot \text{op}(A) \]

where \( \alpha \) is a scalar, \( B \) is an \( M \times N \) matrix, \( A \) is a unit, or non-unit, upper or lower triangular matrix, and \( \text{op}(A) \) is either \( \text{op}(A) = A \) or \( \text{op}(A) = A' \).

Parameters

\texttt{SIDE}\n
On entry, \texttt{SIDE} specifies whether \( \text{op}(A) \) multiplies \( B \) from the left or right as follows:

\texttt{SIDE} = 'L' or 'l' \quad \Rightarrow \quad B := \alpha \cdot \text{op}(A) \cdot B

\texttt{SIDE} = 'R' or 'r' \quad \Rightarrow \quad B := \alpha \cdot B \cdot \text{op}(A)

Unchanged on exit.

\texttt{UPLO}\n
On entry, \texttt{UPLO} specifies whether the matrix \( A \) is an upper or lower triangular matrix as follows:

\texttt{UPLO} = 'U' or 'u' \quad \Rightarrow \quad A \text{ is an upper triangular matrix.}

\texttt{UPLO} = 'L' or 'l' \quad \Rightarrow \quad A \text{ is a lower triangular matrix.}

Unchanged on exit.

\texttt{TRANSA}\n
On entry, \texttt{TRANSA} specifies the form of \( \text{op}(A) \) to be used in the matrix multiplication as follows:

\texttt{TRANSA} = 'N' or 'n' \quad \Rightarrow \quad \text{op}(A) = A

\texttt{TRANSA} = 'T' or 't' \quad \Rightarrow \quad \text{op}(A) = A'

\texttt{TRANSA} = 'C' or 'c' \quad \Rightarrow \quad \text{op}(A) = A'

Unchanged on exit.

\texttt{DIAG}\n
On entry, \texttt{DIAG} specifies whether or not \( A \) is unit triangular as follows:

\texttt{DIAG} = 'U' or 'u' \quad \Rightarrow \quad A \text{ is assumed to be unit triangular.}

\texttt{DIAG} = 'N' or 'n' \quad \Rightarrow \quad A \text{ is not assumed to be unit triangular.}

Unchanged on exit.

\texttt{M}\n
On entry, \( M \) specifies the number of rows of \( B \); \( M \) must be at least 0; unchanged on exit.

\texttt{N}\n
On entry, \( N \) specifies the number of columns of \( B \); \( N \) must be at least 0; unchanged on exit.

\texttt{ALPHA}\n
On entry, \texttt{ALPHA} specifies the scalar \( \alpha \). When \( \alpha = 0 \) then \( A \) is not referenced and \( B \) need not be set before entry; unchanged on exit.
A

An array of dimension \((LDA, k)\), where \(k = M\) when \(SIDE = 'L'\) or \('l'\) and is \(N\) when \(SIDE = 'R'\) or \('r'\); on entry with \(UPLO = 'U'\) or \('u'\), the leading \(k\) by \(k\) upper triangular part of the array \(A\) must contain the upper triangular matrix and the strictly lower triangular part of \(A\) is not referenced; on entry with \(UPLO = 'L'\) or \('l'\), the leading \(k\) by \(k\) lower triangular part of the array \(A\) must contain the lower triangular matrix and the strictly upper triangular part of \(A\) is not referenced. When \(DIAG = 'U'\) or \('u'\), the diagonal elements of \(A\) are not referenced either, but are assumed to be unity; unchanged on exit.

\(LDA\)

On entry, \(LDA\) specifies the first dimension of \(A\) as declared in the calling (sub) program. When \(SIDE = 'L'\) or \('l'\) then \(LDA\) must be at least max\((1, M)\), when \(SIDE = 'R'\) or \('r'\) then \(LDA\) must be at least max\((1, N)\); unchanged on exit.

\(B\)

An array of dimension \((LDB, N)\); on entry, the leading \(M\) by \(N\) part of the array \(B\) must contain the matrix \(B\), and on exit is overwritten by the transformed matrix.

\(LDB\)

On entry, \(LDB\) specifies the first dimension of \(B\) as declared in the calling (sub) program; \(LDB\) must be at least max\((1, M)\); unchanged on exit.

---

**STRSM, DTRSM, CTRSM, or ZTRSM Subroutine**

**Purpose**

Solves certain matrix equations.

**Library**

BLAS Library (libblas.a)

**FORTRAN Syntax**

```
SUBROUTINE STRSM(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)
CHARACTER*1 SIDE, UPLO, TRANSA, DIAG
INTEGER M, N, LDA, LDB
REAL ALPHA
REAL A(LDA,*), B(LDB,*)
```

```
SUBROUTINE DTRSM(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)
CHARACTER*1 SIDE, UPLO, TRANSA, DIAG
INTEGER M, N, LDA, LDB
DOUBLE PRECISION ALPHA
DOUBLE PRECISION A(LDA,*), B(LDB,*)
```

```
SUBROUTINE CTRSM(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)
CHARACTER*1 SIDE, UPLO, TRANSA, DIAG
INTEGER M, N, LDA, LDB
COMPLEX ALPHA
COMPLEX A(LDA,*), B(LDB,*)
```

```
SUBROUTINE ZTRSM(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)
CHARACTER*1 SIDE, UPLO, TRANSA, DIAG
INTEGER M, N, LDA, LDB
COMPLEX*16 ALPHA
COMPLEX*16 A(LDA,*), B(LDB,*)
```

**Description**

The STRSM, DTRSM, CTRSM, or ZTRSM subroutine solves one of the matrix equations:

- \(\text{op}( A ) \times X = \alpha B\)
- \(X \times \text{op}( A ) = \alpha B\)
where alpha is a scalar, X and B are M by N matrices, A is a unit, or non-unit, upper or lower triangular matrix, and \( \text{op}(A) \) is either \( \text{op}(A) = A \) or \( \text{op}(A) = A' \). The matrix X is overwritten on B.

Parameters

**SIDE**

On entry, **SIDE** specifies whether \( \text{op}(A) \) appears on the left or right of X as follows:

- **SIDE** = 'L' or 'l'
  
  \( \text{op}(A) \times X = \alpha \times B \)

- **SIDE** = 'R' or 'r'
  
  \( X \times \text{op}(A) = \alpha \times B \)

Unchanged on exit.

**UPLO**

On entry, **UPLO** specifies whether the matrix A is an upper or lower triangular matrix as follows:

- **UPLO** = 'U' or 'u'
  
  A is an upper triangular matrix.

- **UPLO** = 'L' or 'l'
  
  A is a lower triangular matrix.

Unchanged on exit.

**TRANSA**

On entry, **TRANSA** specifies the form of \( \text{op}(A) \) to be used in the matrix multiplication as follows:

- **TRANSA** = 'N' or 'n'
  
  \( \text{op}(A) = A \)

- **TRANSA** = 'T' or 't'
  
  \( \text{op}(A) = A' \)

- **TRANSA** = 'C' or 'c'
  
  \( \text{op}(A) = A' \)

Unchanged on exit.

**DIAG**

On entry, **DIAG** specifies whether or not A is unit triangular as follows:

- **DIAG** = 'U' or 'u'
  
  A is assumed to be unit triangular.

- **DIAG** = 'N' or 'n'
  
  A is not assumed to be unit triangular.

Unchanged on exit.

**M**

On entry, **M** specifies the number of rows of B; M must be at least 0; unchanged on exit.

**N**

On entry, **N** specifies the number of columns of B; N must be at least 0; unchanged on exit.

**ALPHA**

On entry, **ALPHA** specifies the scalar alpha. When alpha is 0 then A is not referenced and B need not be set before entry; unchanged on exit.

**A**

An array of dimension (LDA, k), where k is M when SIDE = 'L' or 'l' and is N when SIDE = 'R' or 'r'. On entry with **UPLO** = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced; on entry with **UPLO** = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. When **DIAG** = 'U' or 'u', the diagonal elements of A are not referenced, but are assumed to be unity; unchanged on exit.

**LDA**

On entry, **LDA** specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l', LDA must be at least \( \max(1, M) \); when SIDE = 'R' or 'r', LDA must be at least \( \max(1, N) \); unchanged on exit.

**B**

An array of dimension (LDB, N); on entry, the leading M by N part of the array B must contain the right-hand side matrix B, and on exit is overwritten by the solution matrix X.

**LDB**

On entry, **LDB** specifies the first dimension of B as declared in the calling (sub) program. LDB must be at least \( \max(1, M) \); unchanged on exit.
Appendix A. Base Operating System Error Codes for Services That Require Path-Name Resolution

The following errors apply to any service that requires path name resolution:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EACCES</td>
<td>Search permission is denied on a component of the path prefix.</td>
</tr>
<tr>
<td>EFAULT</td>
<td>The Path parameter points outside of the allocated address space of the process.</td>
</tr>
<tr>
<td>EIO</td>
<td>An I/O error occurred during the operation.</td>
</tr>
<tr>
<td>ELOOP</td>
<td>Too many symbolic links were encountered in translating the Path parameter.</td>
</tr>
<tr>
<td>ENAMETOOLONG</td>
<td>A component of a path name exceeded 255 characters and the process has the DisallowTruncation attribute (see the ulimit subroutine) or an entire path name exceeded 1023 characters.</td>
</tr>
<tr>
<td>ENOENT</td>
<td>A component of the path prefix does not exist.</td>
</tr>
<tr>
<td>ENOENT</td>
<td>A symbolic link was named, but the file to which it refers does not exist.</td>
</tr>
<tr>
<td>ENOENT</td>
<td>The path name is null.</td>
</tr>
<tr>
<td>ENOTDIR</td>
<td>A component of the path prefix is not a directory.</td>
</tr>
<tr>
<td>ESTALE</td>
<td>The root or current directory of the process is located in a virtual file system that is unmounted.</td>
</tr>
</tbody>
</table>

Related Information

List of File and Directory Manipulation Services
### Appendix B. ODM Error Codes

When an ODM subroutine is unsuccessful, a value of -1 is returned and the `odmererrno` variable is set to one of the following values:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODMI_BAD_CLASSNAME</td>
<td>The specified object class name does not match the object class name in the file. Check path name and permissions.</td>
</tr>
<tr>
<td>ODMI_BAD_CLXNNNAME</td>
<td>The specified collection name does not match the collection name in the file.</td>
</tr>
<tr>
<td>ODMI_BAD_CRIT</td>
<td>The specified search criteria is incorrectly formed. Make sure the criteria contains only valid descriptor names and the search values are correct. <a href="#">For information on qualifying criteria, see “Understanding ODM Object Searches” in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.</a></td>
</tr>
<tr>
<td>ODMI_BAD_LOCK</td>
<td>Cannot set a lock on the file. Check path name and permissions.</td>
</tr>
<tr>
<td>ODMI_BAD_TIMEOUT</td>
<td>The time-out value was not valid. It must be a positive integer.</td>
</tr>
<tr>
<td>ODMI_BAD_TOKEN</td>
<td>Cannot create or open the lock file. Check path name and permissions.</td>
</tr>
<tr>
<td>ODMI_CLASS_DNE</td>
<td>The specified object class does not exist. Check path name and permissions.</td>
</tr>
<tr>
<td>ODMI_CLASS_EXISTS</td>
<td>The specified object class already exists. An object class must not exist when it is created.</td>
</tr>
<tr>
<td>ODMI_CLASS_PERMS</td>
<td>The object class cannot be opened because of the file permissions.</td>
</tr>
<tr>
<td>ODMI_CLXNMAGICNO_ERR</td>
<td>The specified collection is not a valid object class collection.</td>
</tr>
<tr>
<td>ODMI_FORK</td>
<td>Cannot fork the child process. Make sure the child process is executable and try again.</td>
</tr>
<tr>
<td>ODMI_INTERNAL_ERR</td>
<td>An internal consistency problem occurred. Make sure the object class is valid or contact the person responsible for the system.</td>
</tr>
<tr>
<td>ODMI_INVALID_CLASS</td>
<td>The specified file is not an object class.</td>
</tr>
<tr>
<td>ODMI_INVALID_CLXN</td>
<td>Either the specified collection is not a valid object class collection or the collection does not contain consistent data.</td>
</tr>
<tr>
<td>ODMI_INVALID_PATH</td>
<td>The specified path does not exist on the file system. Make sure the path is accessible.</td>
</tr>
<tr>
<td>ODMI_LINK_NOT_FOUND</td>
<td>The object class that is accessed could not be opened. Make sure the linked object class is accessible.</td>
</tr>
<tr>
<td>ODMI_LOCK_BLOCKED</td>
<td>Cannot grant the lock. Another process already has the lock.</td>
</tr>
<tr>
<td>ODMI_LOCK_ENV</td>
<td>Cannot retrieve or set the lock environment variable. Remove some environment variables and try again.</td>
</tr>
<tr>
<td>ODMI_LOCK_ID</td>
<td>The lock identifier does not refer to a valid lock. The lock identifier must be the same as what was returned from the <code>odm_lock</code> subroutine.</td>
</tr>
<tr>
<td>ODMI_MAGICNO_ERR</td>
<td>The class symbol does not identify a valid object class.</td>
</tr>
<tr>
<td>ODMI_MALLOC_ERR</td>
<td>Cannot allocate sufficient storage. Try again later or contact the person responsible for the system.</td>
</tr>
<tr>
<td>ODMI_NO_OBJECT</td>
<td>The specified object identifier did not refer to a valid object.</td>
</tr>
<tr>
<td>ODMI_OPEN_ERR</td>
<td>Cannot open the object class. Check path name and permissions.</td>
</tr>
<tr>
<td>ODMI_OPEN_PIPE</td>
<td>Cannot open a pipe to a child process. Make sure the child process is executable and try again.</td>
</tr>
<tr>
<td>ODMI_PARAMS</td>
<td>The parameters passed to the subroutine were not correct. Make sure there are the correct number of parameters and that they are valid.</td>
</tr>
<tr>
<td>ODMI_READ_ONLY</td>
<td>The specified object class is opened as read-only and cannot be modified.</td>
</tr>
<tr>
<td>ODMI_READ_PIPE</td>
<td>Cannot read from the pipe of the child process. Make sure the child process is executable and try again.</td>
</tr>
<tr>
<td>ODMI_TOOMANYCLASSES</td>
<td>Too many object classes have been accessed. An application can only access less than 1024 object classes.</td>
</tr>
<tr>
<td>ODMI_UNLINKCLASS_ERR</td>
<td>Cannot remove the object class from the file system. Check path name and permissions.</td>
</tr>
<tr>
<td>ODMI_UNLINKCLXN_ERR</td>
<td>Cannot remove the object class collection from the file system. Check path name and permissions.</td>
</tr>
<tr>
<td>ODMI_UNLOCK</td>
<td>Cannot unlock the lock file. Make sure the lock file exists.</td>
</tr>
</tbody>
</table>
Related Information

List of ODM Commands and Subroutines in AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs.
Appendix C. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Dept. LRAS/Bldg. 003
11400 Burnet Road
Austin, TX 78758-3498
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

© Copyright IBM Corp. 1994, 2005
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

**Trademarks**

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

- AFS
- AIX
- AIX 5L
- IBM

UNIX is a registered trademark of The Open Group in the United States and other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be the trademarks or service marks of others.
Index

Special characters
_lazySetErrorHandler subroutine 634
_showstring subroutine 676
_sync_cache_range subroutine 354

Numerics
8-bit character capability 638

A
absolute values
finding index of element with maximum value 719
access control information
retrieving 312
access control subroutines
fstatacl 312
revoke 58
statacl 312
accounting subroutines
rmproj 61
rmprojdb 62
addch subroutine 573
addstr subroutine 574
alarm signals
beeping 580
flashing 607
alphasort subroutine 124
alternate stack 214
argument formatting
vfscanf 480
vscanf 480
vsscanf 480
asynchronous serial data line
sending breaks on 391
atoi subroutine 341
attroff subroutine 576
attron subroutine 578
attrset subroutine 578
authentication database
opening and closing 184
authentication subroutines
endpwdb 184
enduserdb 189
setpwdb 184
setuserdb 189
tcb 384

B
backspace character
returning 606
baudrate subroutine 579
beep subroutine 580
Berkeley Compatibility Library
subroutines
rand_r 7

binary trees, manipulating 458
BLAS matrix-matrix operations 761
BLAS matrix-matrix subroutines 755, 756, 758, 760,
763, 764, 766, 768
BLAS matrix-vector subroutines 723, 724, 726, 727,
729, 730, 731, 732, 733, 735, 737, 738, 740, 742,
744, 745, 746, 747, 748, 749, 750, 751, 752, 754
BLAS vector-vector functions 711, 712, 717, 718, 719,
720
BLAS vector-vector subroutines 713, 714, 715, 716,
718, 721, 722
box subroutine 581
buffers
assigning to streams 163
bytes
copying 346

C
carriage return 648
CAXPY subroutine 713
cbox subroutine 581
cboxalt subroutine 581
CBREAK mode 585
cbreak subroutine 585
CCOPY subroutine 715
CDOTC function 711
CDOTU function 712
CGBMV subroutine 724
CGEMM subroutine 755
CGEMV subroutine 723
CGERC subroutine 745
CGERU subroutine 745
change color definition 625
change color-pair definition 626
change terminal capabilities 607
character conversion
wide characters
lowercase to uppercase 414
to double-precision number 503
to long integer 508
to multibyte 510, 517
to tokens 506
to unsigned long integer 511
uppercase to lowercase 413
character data
interpreting 126
reading 126
character manipulation subroutines
vwsprintf 487
character mapping 518
character transliteration 412
characters
adding
lines 629
single characters 573, 628
strings 574
backspace 606
characters (continued)
clearing screen 586, 587
controlling text scrolling 668, 669, 671
deleting 600
dumping strings 676
echoing 602
erasing lines 590, 591, 601
erasing window 605
getting single characters 611
getting strings 616
handling input 638, 648
line-kill 633
placing at cursor location 624
reading formatted input 664
refreshing 698, 700
type ahead 703
typeahead 608
writing 487
writing formatted output 654
charsetID
  wide character 497
CHBMV subroutine 727
CHEMM subroutine 758
CHEMV subroutine 726
CHER subroutine 746
CHER2 subroutine 748
CHER2K subroutine 764
CHERK subroutine 761
CHPMV subroutine 729
CHPR subroutine 747
CHPR2 subroutine 749
clear subroutine 586
clearok subroutine 587
close role database 185
close SMIT ACL database 161
closelog subroutine 372
closelog_r subroutine 375
clrtobot subroutine 590
clrtoeol subroutine 591
code sets
  reading map files 165
color definition 625
color intensity 592
color manipulation 582
color pair 652
color support 620
color-pair definition 626
color, initialize 686
columns
  determining number 674, 690
compare wide character 549
complex dot products
  determining 711, 712
control characters
  specifying 704
control input characters 620
convert wide character 516
converter subroutines
  wcsrtombs 501
copy a window region 593
copy wide character 550
create subwindows 687
cresetty subroutine 661
CROTG subroutine 713
CSCAL subroutine 718
CSROT subroutine 714
CSSCAL subroutine 718
CSWAP subroutine 716
CSYMM subroutine 756
CSYR2K subroutine 763
CSYRK subroutine 760
CTBMV subroutine 735
CTBSV subroutine 740
CTPMV subroutine 737
CTPSV subroutine 742
CTRMM subroutine 766
CTRMV subroutine 733
CTRSMV subroutine 768
CTRSV subroutine 738
current process credentials
  setting 175
current process environment
  setting 178
current processes
  group ID
    setting 169
  suspending 236
  user information 475
current screen
  refreshing 658, 707
current screens
  refreshing 653
curses
  initializing 627
  terminating 604
curses character control subroutines
  _showstring 676
  addch 573
  addstr 574
  clear 586
  clearok 587
  clrtobot 590
  clrtoeol 591
  delch 600
  deleteln 601
  erase 605
  getch 611
  getstr 616
  inch 624
  insch 628
  insertln 629
  meta 638
  mvaddch 573
  mvaddstr 574
  mvdelch 600
  mvgetch 611
  mvgetstr 616
  mvinch 624
  mvinsch 628
  mvscanw 664
  mwaddch 573
  mwaddstr 574
curses character control subroutines (continued)
  mwdelch 600
  mwgetch 611
  mvwgetstr 616
  mvwinch 624
  mvwinsch 628
  mvwscanw 664
  nodelay 648
  scanw 664
  scroll 668
  scrollok 669
  setscrreg 671
  unctrl 704
  waddch 573
  waddstr 574
  wclear 586
  wclrtobot 590
  wclrtoeol 591
  wdelch 600
  wdeleteln 601
  werase 605
  wgetch 611
  wgetstr 616
  winch 624
  winsch 628
  winsertln 629
  wscanw 664
  wsetscreg 671

curses cursor control subroutines
  getyx 619
  leaveok 635
  move 639
  mvcur 639
  wmove 639

curses data structure 666

curses options setting subroutines
  idlok 623
  intrflush 630
  keypad 632
  typeahead 703

curses portability subroutines
  baudrate 579
  erasechar 606
  flushinp 608
  killchar 633

curses subroutine
  getbegyx 610
  getmaxyx 615

curses subroutines
  character locations
    echochar, wechochar, pechochar 603
  endwin 604
  initscr 627
  switching input/output to different terminals 673

curses terminal manipulation subroutines
  cbreak 585
  cresetty 661
  def_prog_mode 595
  def_shell_mode 596
  delay_output 599
  echo 602

curses terminal manipulation subroutines (continued)
  has_ic 621
  has_il 622
  longname 636
  newterm 644
  nl 648
  nocbreak 585
  noecho 602
  nonl 648
  noraw 657
  putp 655
  raw 657
  reset_prog_mode 659
  reset_shell_mode 659
  resetterm 660
  resetty 661
  set_term 673
  setupterm 674
  tgetent 690
  tgetflag 691
  tgetnum 692
  tgetstr 692
  tgoto 693
  tparm 701
  tputs 702

curses video attributes subroutines
  attroff 576
  attron 578
  attrset 578
  beep 580
  flash 607
  standend 684
  standout 684
  vidattr 706
  vidputs 706
  wattroff 576
  wattron 578
  wattset 578
  wstandend 684
  wstandout 684

curses window manipulation subroutines
  box 581
  delwin 602
  douupdate 707
  makenew 637
  mvwin 641
  newpad 642
  newwin 646
  overlay 651
  overwrite 651
  pnoutrefresh 653
  prefresh 653
  refresh 658
  subwin 688
  touchline 698
  touchoverlap 699
  touchwin 700
  wnoutrefresh 707
  wrefresh 658
  moving logical cursor 639
cursor control (continued)
  moving physical cursor 639
  placing cursor 635
  returning logical cursor coordinates 619
cursor coordinates 610
cursor visibility 594

d
D cache 354
DASUM subroutine 718
data
  sorting with quicker-sort algorithms 1
data sorting subroutines
    qsort 1
tdelete 458
tfind 458
tsearch 458
twalk 458
data transmissions
  suspending 386
  waiting for completion 385
data words
  trace 449
databases
  authentication
    opening and closing 184
date
  format conversions 330
date format conversions 343, 496
DAXPY subroutine 713
DCOPY subroutine 715
DDOT function 711
def_prog_mode subroutine 595
def_shell_mode subroutine 596
defect 220643 412
define character mapping 518
delay mode 620
delch subroutine 599
delteeln subroutine 601
delwin subroutine 602
determine terminal color support 620
device driver
  calling 360
device switch tables
    checking entry status 368
DGBMV subroutine 724
DGEMM subroutine 755
DGEMV subroutine 723
DGER subroutine 744
directories
  reading 33
  removing 60
  removing entries 471
  renumbering 55
  scanning contents 124
  sorting contents 124
directory subroutines
  alphasort 124
  readlink 35
directory subroutines (continued)
    rmdir 60
    scandir 124
    symlink 350
    unlink 471
disable terminal capabilities 607
discard lines in windows 609
disk quotas
  manipulating 2
  DNRM2 function 717
dot products
    determining 711, 720
doupdate subroutine 707
drawbox subroutine 581
drawboxalt subroutine 581
DROT subroutine 714
DROTG subroutine 713
DROTM subroutine 721
DROTMG subroutine 722
DSBMV subroutine 731
DSCAL subroutine 718
DSPMV subroutine 732
DSPR subroutine 751
DSPR2 subroutine 754
DSWAP subroutine 716
DSYMM subroutine 756
DSYMV subroutine 730
DSYR subroutine 750
DSYR2 subroutine 752
DSYR2K subroutine 763
DSYRK subroutine 760
DTBMV subroutine 735
DTBSV subroutine 740
DTPMV subroutine 737
DTPSV subroutine 742
DTRMM subroutine 766
DTRMV subroutine 733
DTRSM subroutine 768
DTRSV subroutine 738
dump file, data structure 666
dump file, restore screen 668
DZASUM subroutine 718
DZNRM2 function 717
e
echo subroutine 602
echochar subroutine 603
echoing characters 602
endpwd subroutine 184
endroledb subroutine 185
enduserdb subroutine 189
endwin subroutine 604
equations
  solving systems 738, 740, 742
erase subroutine 605
erasechar subroutine 606
error codes 771
error codes, ODM 773
error handler, install 634
error handling
controlling system logs 372
numbering error message string 327
errorlogging subroutines
closelog 372
openlog 372
setlogmask 372
syslog 372
errologging_r subroutines 375
Euclidean lengths
determining 717
examine state of alternate stack 214
execution control
saving and restoring context 170
execution control subroutines
 equaljmp 170
 setjmp 170
exponential numbers
scalb 122
scalb nf 122
scalb nl 122
scalb 122
scalbf 122
scalbn 122
extended attribute subroutines
geta  166
 remov e a 53
 statea 314
extended curses
initializing 627
extended curses character control subroutines
_showstring 676
getch 611
 inch 624
 insch 628
meta 638
mvgetch 611
mvinch 624
mvinsch 628
mvscanw 664
mvwgetch 611
mvwinch 624
mvwinsch 628
mvwscanw 664
printw 654
 scanw 664
 scroll 668
scrollok 669
wgetch 611
 winch 624
winsch 628
wscanw 664
extended curses options setting subroutines
idlok 623
intrflush 630
extended curses portability subroutines
baudrate 579
erasechar 606
flushinp 608
killchar 633
extended curses subroutines
initscr 627
extended curses terminal manipulation subroutines
delay_output 599
has_ic 621
has il 622
newterm 644
putp 655
resetterm 660
set_term 673
setupterm 674
tgentent 690
tgetflag 691
tgetnum 692
tparm 701
extended curses video attributes subroutines
attroff 576
attron 578
attrset 578
 standend 685
standout 685
vidputs 706
wattroff 576
 wattron 578
wattrset 578
wstandend 684
wstandout 685
extended curses window manipulation subroutines
box 581
cbox 581
cboxalt 581
delwin 602
doupdate 707
drawbox 581
drawboxalt 581
fullbox 581
makenew 637
mvwin 641
newwin 646
 overlay 651
overwrite 651
superbox 581
superbox1 581
touchline 698
touchoverlap 700
wnoutrefresh 707
F
ffullstat subroutine 319
file access times
setting 476
file creation masks
getting or setting values 466
file descriptors
checking I/O status 139
file modification times
setting 476
file subroutines
ffullstat 319
fstat 319
Index 781
file subroutines (continued)
fstat 319
ftruncate 455
fullstat 319
lstat 319
remove 52
rename 55
stat 319
statx 319
tempnam 411
tmpfile 410
tmpnam 411
truncate 455
umask 466
utime 476
utimes 476
file system information 317
file system subroutines
fstatfs 315
fstatfs64 315
mount 484
quotactl 2
statfs 315
statfs64 315
sync 352
sysconf 355
umount 467
ustat 315
uvmount 467
vmount 484
file systems
manipulating disk quotas 2
mounting 484
returning statistics 315
unmounting 467
updating 352
file, input/output 665
files
changing length of regular 455
constructing names for temporary 411
creating symbolic links 350
creating temporary 410
deleting 52
providing status information 319
reading 29
removing 52
renaming 55
revoking access 58
writing to 554
find wide character 548
find wide character substring 503
flash subroutine 607
flow control
performing 386
flushing
typeahead characters 608
flushinp subroutine 608
foreground process group IDs
getting 390
setting 394
formatted input
converting 126
fscanf subroutine 126
fstat subroutine 318
fstat64x subroutine 318
fstatfs subroutine 318
fstatfs64 subroutine 318
fstatvfs subroutine 317
fstatvfs64 subroutine 317
fstatx subroutine
described 318
ftruncate subroutine 455
fullbox subroutine 581
fullstat subroutine 319

G
gamma subroutines
tgamma 397
tgammaf 397
tgammal 397
get capabilities, terminfo 694
get key name 631
get terminals numeric value 696
get terminals string capability 697
get XTI variables 426
get_wctype subroutine 518
getbegyx subroutine 610
getch subroutine 611, 638, 648
getmaxyx subroutine 615
getstr subroutine 616
getyx macro 619
Givens plane rotations
constructing 713
Givens transformations
applying 721
constructing 722
gsignal subroutine 311
gtty subroutine 345

H
half-delay mode 620
has_ic subroutine 621
has_il subroutine 622
Hermitian operations
performing rank 1 746, 747
performing rank 2 748, 749
performing rank 2k 764
performing rank k 761
highlight mode 684
hook words
trace 449
hyperbolic functions
computing 234
hyperbolic sine subroutines
sinhf 234
hyperbolic tangent subroutines
tanhf 383
I
I cache 354
I/O asynchronous subroutines
  select 139
I/O low-level subroutines 29, 554
  readvx 29
  readx 29
  writevx 554
  writex 554
I/O stream subroutines
  fscanf 126
  scanf 126
  setbuf 163
  setbuffer 163
  setlinebuf 163
  setvbuf 163
  ungetc 470
  ungetwc 470
  wsscanf 126
I/O terminal subroutines
  gtty 345
  isatty 462
  stty 345
  tcdrain 385
  tcflow 386
  tcflush 388
  tcgetattr 389
  tcgetattrpgpr 390
  tcsendbreak 391
  tcsetattr 392
  tcsetpgpr 394
  termdef 395
  ttylock 460
  ttylocked 460
  tyname 462
  ttyset 462
  ttyunlock 460
  ttywait 460
ICAMAX subroutine 719
IDAMAX subroutine 719
idlok subroutine 623
idxpg4 323
inch subroutine 624
index subroutine 333
initialize color 686
initscr subroutine 627
initstate subroutine 8
input streams
  pushing single character into 470
insch subroutine 628
insert-character capability 621
insert-line capability 622
insert/delete line option 623
insertln subroutine 629
interval timers
  releasing 51
intrflush subroutine 630
ISAMAX subroutine 719
isatty subroutine 462
IZAMAX subroutine 719

J
JFS
  manipulating disk quotas 2

K
kernel configurations
  customizing 358
kernel extension modules
  loading 372
kernel extensions
  loading 364
kernel object files
  determining status 369
  invoking 361
  unloading 367
kernel parameters
  setting 370
key name 631
keypad
  enabling 632
  keypad subroutine 632
  killchar subroutine 633
label name, return 680
lazy loading runtime system 634
LC_ALL environment variable 172
LC_COLLATE category 172
LC_CTYPE category 172
LC_MESSAGES category 172
LC_MONETARY category 172
LC_NUMERIC category 172
LC_TIME category 172
leaveok subroutine 635
line-kill character 633
lines
  adding 629
  determining number 674, 690
  erasing 590, 591, 601
links
  creating symbolic 350
  reading contents of symbolic 35
locale subroutines
  rmatch 64
  setlocale 171
locales
  changing or querying 171
  response matching 64
locking functions
  controlling tty 460
logical cursor 619, 639
long integers, converting
  from character strings 341
  from wide-character strings 508
long numeric data 190
longjmp subroutine 170
longname subroutine 636
lowercase characters
  converting from uppercase 413
  converting to uppercase 414
lstat subroutine 318
lstat64x subroutine 318
minicurses subroutines (continued)
  flushinp 608
  getch 611
  m_initscr 627
monetary strings 328
mount subroutine 484
mounted file systems
  returning statistics 315
move subroutine 639
multibyte characters
  converting from wide 510, 517
mvaddch subroutine 573
mvaddstr subroutine 574
mvcur subroutine 639
mvdelch subroutine 600
mvgetch subroutine 611
mvgettext subroutine 616
mvinsch subroutine 624
mvinscr subroutine 628
mvprintw subroutine 654
mvscanw subroutine 664
mwaddch subroutine 573
mwaddstr subroutine 574
mwdelch subroutine 600
mwgetch subroutine 611
mwgettext subroutine 616
mwinc subroutine 624
mwincsch subroutine 628
mwprintw subroutine 654
mwscanw subroutine 664
M
m_initscr subroutine 627
makenew subroutine 637
mapped files
  attaching to process 194
mapping, character 518
matrices
  performing matrix-matrix operations with
    general matrices 755
    Hermitian matrices 758
    symmetric matrices 756
    triangular matrices 766
  performing matrix-vector operations with
    general banded matrices 724
    general matrices 723
    Hermitian band matrices 727
    Hermitian matrices 726
    packed Hermitian matrices 729
    packed symmetric matrices 732
    packed triangular matrices 737
    symmetric band matrices 731
    symmetric matrices 730
    triangular band matrices 735
    triangular matrices 733
  solving equations 768
memory
  freeing 554
memory management
  activating paging or swapping 347, 348
  controlling shared memory operations 197
  returning paging device status 349
  returning shared memory segments 201
memory management subroutines
  shmat 194
  shmatl 197
  shmct 200
  shmdt 201
  swapoff 347
  swapon 348
  swapoff 349
memory mapping
  attaching segment or file to process 194
message queues
  checking I/O status 139
meta subroutine 638
minicurses
  initializing 627
minicurses subroutines
  attrset 578
  baudrate 579
  erasechar 606
  numbers
    generating
      pseudo-random 6
      random 6, 8
numerical data
  generating pseudo-random numbers 7
numerical manipulation subroutines
  atoi 341
  initstate 8
  rand 6
  random 8
  rint 59
  rint 59
  sqrt 120
  scalb 122
N
new-line character 648
newpad subroutine 642
newterm subroutine 644
newwin subroutine 646
nl subroutine 648
no timeout mode 649
nocbreak subroutine 585
nodelay subroutine 648
noecho subroutine 602
nonl subroutine 648
noraw subroutine 657
nsleep subroutine 236
numbers
  generating
    pseudo-random 6
    random 6, 8
numerical data
  generating pseudo-random numbers 7
numerical manipulation subroutines
  atoi 341
  initstate 8
  rand 6
  random 8
  rint 59
  rint 59
  sqrt 120
  scalb 122

784  Technical Reference, Volume 2: Base Operating System and Extensions
numerical manipulation subroutines (continued)
  setstate 8
  sgetl 190
  sinh 234
  sinl 233
  sputl 190
  sqrt 278
  sqrl 278
  srand 6
  srand 8
  strtold 337
  strtold 337
  strtol 341
  tan 382
  tanh 383
  tanl 382
  watof 563
  watoi 564
  watol 564
  wstrtod 563
  wstrtol 564

O

object file access subroutines
  sgetl 190
  sputl 190

object file subroutines
  unload 473

object files
  unloading 473

Obtaining high-resolution elapsed time
  read_real_time or time_base_to_time 36

ODM error codes 773
open role database 185
open SMIT ACL database 161
openlog subroutine 372
openlog_r subroutine 375
operating system
  customizing configurations 358
  identifying 469
output
  waiting for completion 385
overlay subroutine 651
overwrite subroutine 651

P

paging memory
  activating 347, 348
  returning information on devices 349
parameter lists
  handling variable-length 477
parameter structures
  copying into buffers 363, 364
path name
  resolve 38
path-name resolution 771
pechochar subroutine 603

performance data from remote kernels 122
physical cursor 639
plane rotations
  applying 714
pnoutrefresh subroutine 653
prefresh subroutine 653
print formatted output 487
printf subroutine 654
printw subroutine 654
process credentials
  setting 175
process environments
  setting 178
process group IDs
  returning 390
  setting 167, 182, 186, 394
  supplementary IDs
    setting 169
process identification
  current operating system name 469
process initiation
  restarting system 39
process priorities
  setting scheduled priorities 183
  yielding to higher priorities 570
process resource allocation
  setting and getting user limits 464
process signals
  blocked signal sets
    changing 228
    returning 218
  changing subroutine restart behavior 217
  enhancement and management 222
  handling system-defined exceptions 203
  implementing software signal facility 311
  manipulating signal sets 215
  sending to executing program 5
  signal masks
    replacing 228
    saving or restoring 226
    setting 219
  specifying action upon delivery 203
  stacks
    defining alternate 227
    saving or restoring context 226
process subroutines (security and auditing)
  setegid 167
  seteuid 187
  setgid 167
  setgidx 167
  setgroups 169
  setpcred 175
  setpenv 178
  setregid 167
  setreuid 187
  setregid 167
  setreuid 187
  setuid 187
  setuserid 187
  system 381
  usinfo 475
process user IDs
  setting 187
processes
  handling user information 475
  suspending 236, 488, 491
processes subroutines
  gsignal 311
  raise 5
  reboot 39
  semctl 153
  semget 156
  semop 158
  setpgid 182
  setpgrp 182
  setpri 183
  setsid 186
  sigaddset 215
  sigblock 219
  sigdelset 215
  sigemptyset 215
  sigfillset 215
  sighold 222
  sigignore 222
  siginterrupt 217
  sigismember 215
  siglongjmp 226
  sigpause 228
  sigpending 218
  sigprocmask 219
  sigreise 222
  sigset 222
  sigsetjmp 226
  sigsetmask 219
  sigstack 227
  sigsuspend 228
  ssigmal 464
  unamex 469
  wait 488
  wait3 488
  waitid 491
  waitpid 488
  yield 570
program mode 659
pseudo-random numbers
  generating 6
pthread_kill subroutine 5
push character to input queue 705
putp subroutine 655

Q
  qsort subroutine 1
  queues
    discarding data 388
    quotact1 subroutine 2

R
  ra_attachrset Subroutine 10
  ra_detachrset Subroutine 12
  ra_exec Subroutine 14
  ra_fork Subroutine 16
  ra_free_attachinfo subroutine 18
  ra_get_attachinfo subroutine 19
  ra_getrset Subroutine 21
  ra_mmap subroutine 22
  ra_mmapv subroutine 22
  raise subroutine 5
  rand subroutine 6
  rand_r subroutine 7
  random numbers
    generating 6, 8
  random subroutine 8
  rank 1 operations 744, 745
  raw mode 657
  raw subroutine 657
  re_comp subroutine 40
  re_exec subroutine 40
  re-initializest terminal structures 661
  read operations
    from a file 29
  read subroutine 29
  read_real_time Subroutine 36
  readdir_r subroutine 33
  readlink subroutine 35
  readv subroutine
    described 29
  readdirx subroutine 29
  readx subroutine
    described 29
  realpath subroutine 38
  reboot subroutine 39
  receive data unit 418
  reception of data
    suspending 386
  reciprocals of square roots
    computing 120
  refresh subroutine 658
  refreshing
    characters 698, 700
    current screen 653, 658, 707
    terminal 653, 658
    windows 699, 708
  regcmp subroutine 41
  regcmp subroutine 44
  regerror subroutine 46
  regex subroutine 41
  regex subroutine 47
  regfree subroutine 50
  regular expression subroutines
    regcmp 41
    regcmp 44
    regerror 46
    regex 41
    regex 47
    regfree 50
regular expressions
  comparing 47
  compiling 41, 44
  error messages 46
  freeing memory 50
  matching 41
regular files
  changing length 455
release indication
  user data 415
reltimerid subroutine 51
remainder subroutine 52
remainder subroutines
  remquo 54
  remquof 54
  remquol 54
Remainder subroutines
  remainder 52
  remainderf 52
  remainderl 52
remainderf subroutine 52
remainderl subroutine 52
remote hosts
  rstat subroutine 122
Remote Statistics Interface subroutines
  RsiChangeFeed 68
  RsiChangeHotFeed 69
  RsiClose 70
  RsiCreateStatSet 72
  RsiDelSetHot 73
  RsiDelSetStat 74
  RsiFirstCx 75
  RsiFirstStat 77
  RsiGetHotItem 78
  RsiGetRawValue 80
  RsiGetValue 81
  RsiInit 83
  RsiInstantiate 84
  RsiMainLoop 87
  RsiNextCx 88
  RsiNextStat 89
  RsiOpen 90
  RsiPathAddSetStat 93
  RsiPathGetCx 94
  RsiStartFeed 95
  RsiStartHotFeed 96
  RsiStatGetPath 97
  RsiStopHotFeed 100
remove subroutine 52
removeea subroutine 53
remquo subroutine 54
remquof subroutine 54
remquol subroutine 54
rename subroutine 55
replace lines in windows 609
reserve a screen line 662
reset_malloc_log subroutine 57
reset_prog_mode subroutine 659
reset_shell_mode subroutine 659
resetterm subroutine 660
resetty subroutine 661
Resource Set APIs
  ra_attachrset 10
  ra_detachrset 12
  ra_exec 14
  ra_fork 16
  ra_free_attchinfo 18
  ra_get_attchinfo 19
  ra_getset 21
  rs_alloc 101
  rs_discardname 102
  rs_free 103
  rs_getassociativity 104
  rs_getinfo 105
  rs_getnameattr 106
  rs_getnamedset 108
  rs_getpartition 109
  rs_getread 110
  rs_init 111
  rs_numrads 112
  rs_op 113
  rs_registername 115
  rs_setnameattr 117
  rs_setpartition 119
restore soft function key 682
restore virtual screen 668
retrieves information from terminfo 597
return color intensity 592
return file system information 317
return label, soft label 680
return window size 615
returns color to color pair 652
revoke subroutine 58
rindex subroutine 333
rint subroutine 59
rintf subroutine 59
rintl subroutine 59
ripoffline subroutine 662
rmdir subroutine 60
rmproj subroutine 61
rmprojdb subroutine 62
round subroutine 64
roundf subroutine 64
roundl subroutine 64
rounding numbers
  rintf 59
  rintl 59
  round 64
  roundf 64
  roundl 64
  trunc 455
  truncf 455
  truncl 455
roundl subroutine 64
rpmatch subroutine 64
rs_alloc Subroutine 101
rs_discardname Subroutine 102
rs_free Subroutine 103
rs_getassociativity Subroutine 104
rs_getinfo Subroutine 105
rs_getnameattr Subroutine 106
rs_getnamedset Subroutine 108
rs_getpartition Subroutine 109
rs_getrad Subroutine 110
rs_init Subroutine 111
rs_numrads Subroutine 112
rs_op Subroutine 113
rs_registername Subroutine 115
rs_setpartition Subroutine 119
rsqrt subroutine 120
rstat subroutine 122
runtime tunable parameters
    setting 370

S
SASUM subroutine 718
savetty subroutine 663
SAXPY subroutine 713
scalb subroutine 122
scalbin subroutine 122
scalbnf subroutine 122
scalbln subroutine 122
scalbn subroutine 122
scalbf subroutine 122
scalbnl subroutine 122
scandir subroutine 124
scanf subroutine 126, 664
scanw subroutine 664
SCASUM subroutine 718
sched_get_priority_max subroutine 131
sched_get_priority_min subroutine 131
sched_getparam subroutine 132
sched_getscheduler subroutine 133
sched_rr_get_interval subroutine 134
sched_setparam subroutine 135
sched_setscheduler subroutine 136
sched_yield subroutine 138
scheduling policy and priority
    kernel thread 407
SCNRM2 function 717
SCOPY subroutine 715
scr_dump subroutine 665
scr_init subroutine 666
scr_restore subroutine 668
screen line 662
screens
    refreshing 653, 658, 708
scroll subroutine 668
scrollx subroutine 669
SDOT function 711
SDSDOT function 720
select subroutine 139
sem_close subroutine 143
sem_destroy subroutine 144
sem_getvalue subroutine 145
sem_init subroutine 146
sem_open subroutine 147
sem_post subroutine 149
sem_timedwait subroutine 150
sem_trywait subroutine 151
sem_unlink subroutine 152
sem_wait subroutine 151
semaphore identifiers 156
semaphore operations 153, 158
semaphore subroutines
    sem_timedwait 150
semctl subroutine 153
semget subroutine 156
semop subroutine 158
send data 420
serial data lines
    sending breaks on 391
sessions
    creating 186
set blocking or non-blocking read 649
set cursor visibility 594
set terminal variables 670
set wide character 551
set_curterm subroutine 670
set_term subroutine 673
setauthdb subroutine 162
setauthdb_r subroutine 162
setbuf subroutine 163
setbuffer subroutine 163
setscremap subroutine 165
seteia subroutine 166
setegid subroutine 167
seteuid subroutine 187
setgid subroutine 167
setgids subroutine 167
setgroups subroutine 169
set jmp subroutine 170
getlinebuf subroutine 163
setlocale subroutine 171
setlogmask subroutine 372
setlogmask_r subroutine 375
setpgavalue subroutine 174
setpgavalue64 subroutine 174
setpcred subroutine 175
setpenv subroutine 178
setpgrp subroutine 182
setpgid subroutine 182
setpgrp subroutine 182
setpri subroutine 183
setpgid subroutine 184
setregid subroutine 167
setreuid subroutine 187
setregid subroutine 167
setregid subroutine 167
setregid subroutine 185
setreldbg subroutine 185
setreuid subroutine 187
setsockopt subroutine 671
setsid subroutine 186
setstate subroutine 8
setsx subroutine 672
setuid subroutine 187
setuidx subroutine 187
setupterm subroutine 674
setuserdb subroutine 189
setvbuf subroutine 163
SGBMV subroutine 724
SGEMM subroutine 755
SGEMV subroutine 723
SGER subroutine 744
sgetl subroutine 190
shared memory segments
attaching to process 194
detaching 200
operations on 197
returning 201
shell commands
running 381
shell mode 596, 659
shm_open subroutine 191
shm_unlink subroutine 193
shmat subroutine 194
shmctl subroutine 197
shmdt subroutine 200
shmget subroutine 201
short status requests
sending 299, 302
sigaddset subroutine 215
sigaltstack subroutine 214
sigblock subroutine 219
sigdelset subroutine 215
sigemptyset subroutine 215
sigfillset subroutine 215
siginterrupt subroutine 217
sigismember subroutine 215
siglongjmp subroutine 226
signal masks
replacing 228
saving or restoring 226
setting 219
signal stacks
defining alternate 227
saving or restoring context 226
signbit macro 218
sigpause subroutine 228
signpending subroutine 218
sigprocmask subroutine 219
sigqueue subroutine 221
sigrelse subroutine 222
sigset subroutine 222
sigsetjmp subroutine 226
sigsetmask subroutine 219
sigstack subroutine 227
signsuspend subroutine 228
sigtimedwait subroutine 231
sigwait subroutine 232
sigwaitinfo subroutine 231
sin subroutine 233
sine subroutines
sinf 233
sinf subroutine 233
single-byte conversion 516
sinf subroutine 234
sinfh subroutine 234
sinhl subroutine 234
sinl subroutine 233
sleep subroutine 236
slk_attroff subroutine 676
(No text available in the image)
subroutines (continued)

SPMI interface (continued)

spmiFirstVals 254
spmiFreeHotSet 255
spmiFreeStatSet 256
spmiGetCx 257
spmiGetHotSet 258
spmiGetStat 259
spmiGetStatSet 261
spmiGetValue 262
spmilInit 263
spmilInstantiate 265
spmiNextCx 266
spmiNextHot 267
spmiNextHotItem 268
spmiNextStat 270
spmiNextVals 272
spmiNextValue 272
spmiPathAddSetStat 274
spmiPathGetCx 276
spmiPathGetPath 277

subservers 284, 287

substring, wide character 503

subsystems

going getting status 284, 287
returning status 299, 302
sending requests 293, 296
starting 308
stopping 306

subwin subroutine 688

subwindows 687

supplementary process group IDs
setting 169

swab subroutine 346

swapoff subroutine 347

swapon subroutine 348

swapping memory
activating 347, 348
returning information on devices 349

swapqpry subroutine 349

symbolic links
creating 350
reading contents 35

symmlink subroutine 350

symmetric operations
performing rank 1 750, 751
performing rank 2 752, 754
performing rank 2k 763
performing rank k 760

sync subroutine 352

synchronize I cache with D cache 354

syncvfs subroutine 353

SYSCALL operation 360

SYS_CFGDD operation 361

SYS_CFGKMOD operation 361

SYS_GETLPAR_INFO operation 363

SYS_GETPARMS operation 364

SYS_KLOAD operation 364

SYS_KULOAD operation 367

SYS_QDVSW operation 368
SYS_QUERYLOAD operation 369  
SYS_SETPARMS operation 370  
SYS_SINGLELOAD operation 372  
syconf subroutine 355  
sysconfig operations  
  SYS_CFGDD 360  
  SYS_CFGKMOD 361  
  SYS_GETLPAR_INFO 363  
  SYS_GETPARMS 364  
  SYS_KLOAD 364  
  SYS_KULOAD 367  
  SYS_QDVSW 368  
  SYS_QUERYLOAD 369  
  SYS_SETPARMS 370  
  SYS_SINGLELOAD 372  
syconfig subroutine 358  
syslog subroutine 372  
syslog_r subroutine 375  

system limits  
determining values 355  

System Performance Measurement Interface subroutines  
  SpmiAddSetHot 238  
  SpmiCreateHotSet 241  
  SpmiCreateStatSet 242  
  SpmiDdsAddCx 243  
  SpmiDdsDelCx 244  
  SpmiDdsInit 245  
  SpmiDelSetHot 247  
  SpmiDelSetStat 248  
  SpmiExit 250  
  SpmiFirstCx 250  
  SpmiFirstHot 251  
  SpmiFirstStat 252  
  SpmiFirstVals 254  
  SpmiFreeHotSet 255  
  SpmiFreeStatSet 256  
  SpmiGetCx 257  
  SpmiGetHotSet 258  
  SpmiGetStat 259  
  SpmiGetStatSet 261  
  SpmiGetValue 262  
  SpmimInit 263  
  SpmimInstantiate 265  
  SpmiNextCx 266  
  SpmiNextHot 267  
  SpmiNextHotItem 268  
  SpmiNextStat 270  
  SpmiNextVals 272  
  SpmiNextValue 272  
  SpmiPathAddSetStat 274  
  SpmiPathGetCx 276  
  SpmiPathGetStatPath 277  
system subroutine 381  

T  
  t_sndreldata subroutine 423  
  t_sndv subroutine 420  
  t_sndvudata subroutine 424  
  t_sysconf subroutine 426  

tables  
  sorting data 1  
  tan subroutine 382  
  tanf subroutine 382  
  tangent subroutines  
    tanf 382  
  tanh subroutine 383  
  tanhf subroutine 383  
  tanhl subroutine 383  
  tanl subroutine 382  

TCB attributes  
  querying or setting 384  
  tcb subroutine 384  
  tcdrain subroutine 385  
  tcflow subroutine 386  
  tcflush subroutine 388  
  tcgetatt subroutine 389  
  tcgetpgpr subroutine 390  
  tcsendbreak subroutine 391  
  tcsetatt subroutine 392  
  tcsetpgpr subroutine 394  
  tdelete subroutine 458  
  tempnam subroutine 411  

temporary files  
  constructing names 411  
  creating 410  

termcap identifiers  
  returning Boolean entry 691  
  returning numeric entry 692  
  returning string entry 692  

termdef subroutine 395  

terminal attributes  
  getting 389  
  setting 392  

terminal capabilities  
  applying parameters to 693, 701  
  insert-character capability 621  
  insert-line capability 622  

terminal capabilities, disable 607  

terminal color support 620  

terminal manipulation  
  determining number of lines and columns 674, 690  
  echoing characters 602  
  outputting string with padding information 655, 702  
  switching input/output of curses subroutines 673  
  toggling new-line and return translation 648  

terminal modes  
  CBREAK 585  
  program 659  
  raw 657  
  resetting 660  
  saving 595  
  shell 596, 659  

terminal names 462  

terminal numeric capability 696
terminal speed 579
terminal string capability 697
terminal states
getting 345, 389
setting 345, 392
terminal structures 661
terminal variables 670

terminals
beeping 580
delaying output to 599
determining type 462
flashing 607
getting names 462
putting in video attribute mode 706
querying characteristics 395
refreshing 653, 658
setting up 644
verbose name 636

terminfo database 694
test_and_set subroutine 396
tfind subroutine 458
tgamma subroutine 397
tgammaf subroutine 397
tgammal subroutine 397
tgetent subroutine 690
tgetflag subroutine 691
tgetnum subroutine 692
tgetstr subroutine 692
tgoto subroutine 693
thread_self subroutine 407
thread_setsched subroutine 407

Thread-Safe C Library
subroutines
    rand_r 7
    readdir_r 33
    strtok_r 340

Threads Library
signal, sleep, and timer handling
    raise subroutine 5
    sithreadmask subroutine 229
    sigqueue subroutine 221
    sigtimedwait subroutine 231
    sigwait subroutine 232
    sigwaitinfo subroutine 231
tigetflag subroutine 694
tigetnum subroutine 696
tigetstr subroutine 697
time format conversions 330, 343, 496
time manipulation subroutines
    nsleep 236
    reltimerid 51
    sleep 236
    usleep 236
time stamps
    trace 449
time subroutines
    read_real_time 36
    time_base_to_time 36
time_base_to_time Subroutine 36
    timeout mode 649
timer_create subroutine 398
timer_delete subroutine 400
timer_getcurrentroutine 400
timer_gettime subroutine 400
timer_settime subroutine 400
times subroutine 402
timezone subroutine 404
tmpfile subroutine 410
tmpnam subroutine 411
touchline subroutine 698
touchoverlap subroutine 699
touchwin subroutine 700
towctrans subroutine 412
towlower subroutine 413
towupper subroutine 414
tparm subroutine 701
tputs subroutine 702

trace channels
    halting data collection 452
    recording trace event for 449
    starting data collection 453

trace data
    halting collection 452
    recording 449
    starting collection 453

trace events
    recording 449, 450

trace sessions
    starting 453
    stopping 454

trace subroutines
    trc_reg 445
    trcgen 449
    trcgent 449
    trchook 450
    trchook64 450
    trcoff 452
    trcon 453
    trcstart 453
    trcstop 454
    utrchook 450
    utrhook64 450

transmission of data
    suspending 386
    waiting for completion 385

trc_close subroutine 427
trc_compare subroutine 428
trc_find_first subroutine 428
trc_find_next subroutine 428
trc_free subroutine 432
trc_hkadddset subroutine 433
trc_hkdelset subroutine 433
trc_hkemptyset subroutine 433
trc_hkfillset subroutine 433
trc_hkisset subroutine 433
trc_hookname subroutine 434
trc_libcntl subroutine 435
trc_logininfo subroutine 437
trc_loginfo Subroutine 439
trc_open subroutine 439
trc_perror subroutine 441
trc_pererror subroutine 441
trc_read subroutine 442
update soft labels 681, 682, 684
uppercase characters
  converting from lowercase 414
  converting to lowercase 413
user database
  opening and closing 189
user information
  getting and setting 475
usleep subroutine 236
usrinfo subroutine 475
ustat subroutine 315
utime subroutine 476
utimes subroutine 476
utmp file
  finding current user slot in 463
utrchook subroutine 450
utrchook64 subroutine 450
uvmount subroutine 467

V
varargs macros 477
vectors
  computing constant times vector plus vector 713
  copying X to Y 715
  interchanging X and Y 716
  returning complex dot products 711, 712
  returning dot products 711, 720
  returning sum of absolute values 718
  scaling by constants 718
VFS (Virtual File System)
  mounting 484
  unmounting 467
vfscanf subroutine 480
vfwprintf subroutine 481
vfwscanf subroutine 481
vidattr subroutine 706
video attributes
  alarm signals
    beeping 580
    flashing 607
  highlight mode 685
  putting terminal in specified mode 706
  setting 578
  turning off 576
  turning on 578
vidputs subroutine 706
Virtual File System 484
virtual screen cursor coordinates 618
vmount subroutine 484
vscanf subroutine 480
vsnprintf subroutine 487
vsscanf subroutine 480
vswscanf subroutine 481
vwscanf subroutine 481
vwscanf subroutine 481
vwscanf subroutine 487
vwscanf subroutine 487
waddch subroutine 573
waddstr subroutine 574

W
waddch subroutine 573
waddstr subroutine 574
wait subroutine 488
wait3 subroutine 488
waitid subroutine 491
waitpid subroutine 563
watoi subroutine 564
waitid subroutine 578
wattr off subroutine 590
wclear subroutine 586
wclrtobot subroutine 590
wclrtoeol subroutine 591
wcscat subroutine 493
wcschr subroutine 493
wcscmp subroutine 493
wcscoll subroutine 494
wcscp y subroutine 493
wcscpy subroutine 493
wcscspn subroutine 493
wcsc time subroutine 496
wcsc id subroutine 497
wcslen subroutine 497
wcsncat subroutine 498
wcsncmp subroutine 498
wcst of subroutine 503
wcstomax subroutine 505
wcstok subroutine 506
wcstol subroutine 508
wcstold subroutine 503
wcstoll subroutine 508
wcstombs subroutine 510
wcstoul subroutine 511
wcstou max subroutine 505
wcswcs subroutine 513
wcswidth subroutine 514
wcsxfrm subroutine 515
wctob subroutine 516
wctomb subroutine 517
wctype subroutine 518
wctrans subroutine 518
wcwidth subroutine 520
wdelch subroutine 600
wdeleteln subroutine 601
wechochar subroutine 603
werase subroutine 605
wgetch subroutine 611
wgetstr subroutine 616
wide character format
vfwscanf 481
vswscanf 481
vwscanf 481
wide character output 481
wide character strings
wcstop 503
wide character strings (continued)
wcstop max 505
wcstold 503
wcstou max 505
wide character subroutines
get_wctype 518
towlower 413
towupper 414
ungetc 470
ungetwc 470
wcscat 493
wcschr 493
wcscmp 493
wcscoll 494
wcscpy 493
wcscspn 493
wcst ime 496
wcsc id 497
wcsc len 497
wcscat 498
wcsncmp 498
wcsncpy 498
wcspbrk 499
wcschr 500
wcsspn 502
wcstod 503
wcstok 506
wcstol 508
wcstoll 508
wcstombs 510
wcstoul 511
wcswcs 513
wcswidth 514
wcsxfrm 515
wctomb 517
wctype 518
wcwidth 520
wide character substring 503
wide character to single-byte 516
wide character, memory 548, 549, 550, 551
wide characters
comparing strings 494
converting
from date and time 496
lowercase to uppercase 414
to double-precision number 503
to long integer 508
to multibyte 510, 517
to tokens 506
to unsigned long integer 511
uppercase to lowercase 413
determining display width 514, 520
determining number in string 497
locating character sequences 513
locating single characters 500
obtaining handle for valid property names 518
operations on null-terminated strings 493, 498
pushing into input stream 470
returning charsetID 497
returning number in initial string segment 502
transforming strings to codes 515
window subroutine 624
window coordinates 610
window manipulation
creating structures
 pad 642
 subwindow 688
 window 646
 window buffer 637
drawing boxes 581
marking changed overlap 699
overwriting window 651
refreshing
 characters 698, 700
 current screen 653, 658, 708
 standard screen 708
 terminal 653, 658, 708
 window 699, 707
window size 615
window, copy 593
windows 609
clearing 586, 587
creating 646, 688
deleting 602
erasing 605
moving 641
 refreshing 699, 707
 scrolling 668, 669, 671
 setting standout bit pattern 601
winsch subroutine 628
winsertln subroutine 629
wmemchr subroutine 548
wmemcmp subroutine 549
wmemcpy subroutine 550
wmemmove subroutine 550
wmemmset subroutine 551
wmove subroutine 639
wnoutrefresh subroutine 707
word expansions
 performing 551
wordexp subroutine 551, 554
wordfree subroutine 554
wprintw subroutine 654
wrefresh subroutine 658
write contents of virtual screen 665
write operations
 writing to files 554
write subroutine
described 554
writev subroutine
described 554
writevx subroutine 554
writex subroutine
described 554
wscanw subroutine 664
wsetscreln subroutine 671
wsscanf subroutine 126
wstandend subroutine 684
wstandout subroutine 684
wstring subroutines 560
wstring subroutine
 described 554
wstrcat subroutine 664
wstrcat subroutine 663
wstrcat subroutine 564
x
xcrypt_btoa 565
xcrypt_decrypt subroutine 565
xcrypt_dh subroutine 565
xcrypt_dh_keygen subroutine 565
xcrypt_encrypt subroutine 565
xcrypt_free subroutine 565
xcrypt_hash subroutine 565
xcrypt_hmac subroutine 565
xcrypt_key_setup subroutine 565
xcrypt_mac subroutine 565
xcrypt_malloc subroutine 565
xcrypt_printb subroutine 565
xcrypt_randbuff subroutine 565
xcrypt_sign subroutine 565
xcrypt_verify subroutine 565
XTI variables 426
y
yield subroutine 570
z
ZAXPY subroutine 713
ZCOPY subroutine 715
ZDOTC function 711
ZDOTU function 712
ZDROT subroutine 714
ZDSCAL subroutine 718
ZGEMV subroutine 724
ZGEMM subroutine 755
ZGEMV subroutine 723
ZGER subroutine 745
ZGERU subroutine 745
ZHBMV subroutine 727
ZHEMM subroutine 758
ZHEMV subroutine 726
ZHER subroutine 746
ZHER2 subroutine 748
ZHER2K subroutine 764
ZHERK subroutine 761
ZHPMV subroutine 729
ZHPMV subroutine 729
ZHPR subroutine 747
ZHPR2 subroutine 749
ZROT subroutine 713
ZSCAL subroutine 718
ZSWAP subroutine 716
ZSYMM subroutine 756
ZSYRK subroutine 760
ZTBMV subroutine 735
ZTBV subroutine 740
ZTBSV subroutine 740
ZTPMV subroutine 737
ZTPSV subroutine 742
ZTRMM subroutine 766
ZTRMV subroutine 733
ZTRSM subroutine 768
ZTRSV subroutine 738
Readers’ Comments — We’d Like to Hear from You

AIX 5L Version 5.3
Technical Reference: Base Operating System and Extensions, Volume 2

Publication No. SC23-4914-02

Overall, how satisfied are you with the information in this book?

<table>
<thead>
<tr>
<th>Overall satisfaction</th>
<th>Very Satisfied</th>
<th>Satisfied</th>
<th>Neutral</th>
<th>Dissatisfied</th>
<th>Very Dissatisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How satisfied are you that the information in this book is:

<table>
<thead>
<tr>
<th></th>
<th>Very Satisfied</th>
<th>Satisfied</th>
<th>Neutral</th>
<th>Dissatisfied</th>
<th>Very Dissatisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accurate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy to find</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy to understand</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Well organized</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applicable to your tasks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you?  □ Yes  □ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name

Address

Company or Organization

Phone No.