

AIX 5L Version 5.1



Technical Reference: Base Operating System and Extensions , Volume 2

AIX 5L Version 5.1



Technical Reference: Base Operating System and Extensions , Volume 2

Third Edition (October 2001)

Before using the information in this book, read the general information in "Appendix C. Notices" on page 669.

© Copyright International Business Machines Corporation 1994, 2001. 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	xiii
Who Should Use This Book	xiii
Highlighting	xiii
ISO 9000	xiii
32-Bit and 64-Bit Support for the UNIX98 Specification	xiii
Related Publications	xiv
Trademarks	xiv
Chapter 1. Base Operating System (BOS) Runtime Services (Q-Z)	1
qsort Subroutine	1
quotactl Subroutine	2
raise Subroutine	4
rand or srand Subroutine	5
rand_r Subroutine	6
random, srandom, initstate, or setstate Subroutine	8
read, readx, readv, readvx, or pread Subroutine	9
readdir_r Subroutine	14
readlink Subroutine	15
read_real_time or time_base_to_time Subroutine	17
realpath Subroutine	19
reboot Subroutine	20
re_comp or re_exec Subroutine	21
regcmp or regex Subroutine	22
regcomp Subroutine	25
regerror Subroutine	27
regexec Subroutine	29
regfree Subroutine	32
retimerid Subroutine	33
remove Subroutine	34
rename Subroutine	34
revoke Subroutine	37
rmdir Subroutine	38
rpmatch Subroutine	40
RSiAddSetHot Subroutine	41
RSiChangeFeed Subroutine	43
RSiChangeHotFeed Subroutine	44
RSiClose Subroutine	45
RSiCreateHotSet Subroutine	46
RSiCreateStatSet Subroutine	47
RSiDelSetHot Subroutine	48
RSiDelSetStat Subroutine	50
RSiFirstCx Subroutine	51
RSiFirstStat Subroutine	52
RSiGetHotItem Subroutine	54
RSiGetRawValue Subroutine	56
RSiGetValue Subroutine	57
RSiInit Subroutine	58
RSiInstantiate Subroutine	60
RSiInvite Subroutine	61
RSiMainLoop Subroutine	63
RSiNextCx Subroutine	64
RSiNextStat Subroutine	65
RSiOpen Subroutine	67

RSiPathAddSetStat Subroutine	69
RSiPathGetCx Subroutine	70
RSiStartFeed Subroutine.	71
RSiStartHotFeed Subroutine	72
RSiStatGetPath Subroutine	74
RSiStopFeed Subroutine.	75
RSiStopHotFeed Subroutine	76
rsqrt Subroutine	77
rstat Subroutines.	79
_safe_fetch Subroutine	79
scandir or alphasort Subroutine	80
scanf, fscanf, sscanf, or wscanf Subroutine	82
sched_yield Subroutine	87
select Subroutine	87
semctl Subroutine	90
semget Subroutine	93
semop Subroutine	96
setacldb or endacldb Subroutine	98
setauditostdb or endauditostdb Subroutine	99
setbuf, setvbuf, setbuffer, or setlinebuf Subroutine	100
setcsmap Subroutine.	102
setgid, setrgid, setegid, or setregid Subroutine	103
setgroups Subroutine	105
setjmp or longjmp Subroutine	106
setlocale Subroutine	107
setpcred Subroutine	111
setpenv Subroutine	113
setpgid or setpgrp Subroutine	117
setpri Subroutine	118
setpwdb or endpwdb Subroutine	119
setroldb or endroldb Subroutine	121
setsid Subroutine	122
setuid, setruid, seteuid, or setreuid Subroutine	122
setuserdb or enduserdb Subroutine	124
sgetl or sputl Subroutine	125
shmat Subroutine	126
shmctl Subroutine	130
shmdt Subroutine	132
shmget Subroutine	133
sigaction, sigvec, or signal Subroutine	135
sigaltstack Subroutine	146
sigemptyset, sigfillset, sigaddset, sigdelset, or sigismember Subroutine	147
siginterrupt Subroutine	149
sigpending Subroutine	150
sigprocmask, sigsetmask, or sigblock Subroutine	151
sigset, sighold, sigrelse, or sigignore Subroutine.	153
sigsetjmp or siglongjmp Subroutine	156
sigstack Subroutine	157
sigsuspend or sigpause Subroutine	158
sigthreadmask Subroutine	159
sigwait Subroutine.	161
sin, sinl, cos, cosl, tan, or tanl Subroutine	162
sinh, sinhl, cosh, coshl, tanh, or tanhl Subroutine	163
sleep, nsleep or usleep Subroutine	164
snprintf Subroutine	166
SpmiAddSetHot Subroutine	170

SpmiCreateHotSet	173
SpmiCreateStatSet Subroutine	174
SpmiDdsAddCx Subroutine	175
SpmiDdsDelCx Subroutine	176
SpmiDdsInit Subroutine	178
SpmiDelSetHot Subroutine	179
SpmiDelSetStat Subroutine	181
SpmiExit Subroutine	182
SpmiFirstCx Subroutine	183
SpmiFirstHot Subroutine	184
SpmiFirstStat Subroutine	185
SpmiFirstVals Subroutine	186
SpmiFreeHotSet Subroutine	187
SpmiFreeStatSet Subroutine	188
SpmiGetCx Subroutine	189
SpmiGetHotSet Subroutine	191
SpmiGetStat Subroutine	192
SpmiGetStatSet Subroutine	193
SpmiGetValue Subroutine	195
Spmilnit Subroutine	196
SpmiInstantiate Subroutine	197
SpmiNextCx Subroutine	199
SpmiNextHot Subroutine	200
SpmiNextHotItem Subroutine	201
SpmiNextStat Subroutine	203
SpmiNextVals Subroutine	204
SpmiNextValue Subroutine	205
SpmiPathAddSetStat Subroutine	207
SpmiPathGetCx Subroutine	209
SpmiStatGetPath Subroutine	210
sqrt, sqrtl, or cbrt Subroutine	211
src_err_msg Subroutine	212
src_err_msg_r Subroutine	213
srcrrqs Subroutine	214
srcrrqs_r Subroutine	216
srcsbuf Subroutine	217
srcsbuf_r Subroutine	220
srcsrpy Subroutine	223
srcsrqt Subroutine	226
srcsrqt_r Subroutine	229
srcstat Subroutine	233
srcstat_r Subroutine	235
srcstathdr Subroutine	237
srcstattxt Subroutine	238
srcstattxt_r Subroutine	239
srcstop Subroutine	239
srcstr Subroutine	242
ssignal or gsignal Subroutine	244
statacl or fstatacl Subroutine	245
statfs, fstatfs, or ustat Subroutine	248
statvfs or fstatvfs Subroutine	250
statx, stat, lstat, fstatx, fstat, fullstat, fullstat, stat64, lstat64, or fstat64 Subroutine	251
strcat, strncat, strxfrm, strcpy, strncpy, or strdup Subroutine	255
strcmp, strncmp, strcasecmp, strncasecmp, or strcoll Subroutine	257
strerror Subroutine	259
strfmon Subroutine	260

strftime Subroutine	262
strlen, strchr, strrchr, strpbrk, strspn, strcspn, strstr, or strtok Subroutine	265
strncollen Subroutine.	268
strtol, strtoul, strtoll, strtoull, atol, or atoi Subroutine	269
strptime Subroutine	271
stty or gtty Subroutine	273
swab Subroutine	274
swapoff Subroutine	275
swapon Subroutine	276
swapqry Subroutine	277
symlink Subroutine	278
sync Subroutine	280
_sync_cache_range Subroutine	281
sysconf Subroutine	282
sysconfig Subroutine	286
SYS_CFGDD sysconfig Operation	288
SYS_CFGKMOD sysconfig Operation	289
SYS_GETPARMS sysconfig Operation	290
SYS_KLOAD sysconfig Operation	291
SYS_KULOAD sysconfig Operation	296
SYS_QDVSX sysconfig Operation	297
SYS_QUERYLOAD sysconfig Operation	298
SYS_SETPARMS sysconfig Operation	299
SYS_SINGLELOAD sysconfig Operation	301
syslog, openlog, closelog, or setlogmask Subroutine	301
syslog_r, openlog_r, closelog_r, or setlogmask_r Subroutine	305
sys_parm Subroutine.	308
system Subroutine	310
tcb Subroutine	311
tcdrain Subroutine.	312
tcflow Subroutine	313
tcflush Subroutine	315
tcgetattr Subroutine	316
tcgetpgrp Subroutine.	317
tcsendbreak Subroutine.	318
tcsetattr Subroutine	319
tcsetpgrp Subroutine	321
termdef Subroutine	322
timezone Subroutine	324
thread_post Subroutine	325
thread_post_many Subroutine	326
thread_self Subroutine	327
thread_setsched Subroutine	327
thread_wait Subroutine	329
tmpfile Subroutine.	330
tmpnam or tmpnam Subroutine	331
towctrans Subroutine.	332
towlower Subroutine	333
toupper Subroutine	334
t_rcvreldata Subroutine	335
t_rcvv Subroutine	337
t_rcvvudata Subroutine	339
t_sndv Subroutine	341
t_sndreldata Subroutine.	343
t_sndvudata Subroutine.	345
t_sysconf Subroutine.	347

trcgen or trcgent Subroutine	348
trchook or utrchook Subroutine	349
trcoff Subroutine	350
trcon Subroutine	351
trcstart Subroutine	352
trcstop Subroutine	353
truncate, truncate64, ftruncate, or ftruncate64 Subroutine	354
tsearch, tdelete, tfind or twalk Subroutine	357
ttylock, ttywait, ttyunlock, or ttylocked Subroutine	359
ttyname or isatty Subroutine	360
ttyslot Subroutine	361
ulimit Subroutine	362
umask Subroutine	364
umount or uvmount Subroutine	365
uname or unamex Subroutine	367
ungetc or ungetwc Subroutine	368
unlink Subroutine	369
unload Subroutine	371
unlockpt Subroutine	372
usrinfo Subroutine	373
utimes or utime Subroutine	375
varargs Macros	376
vwprintf, vwprintf Subroutine	379
vmgetinfo Subroutine	379
vmount or mount Subroutine	380
vsprintf Subroutine	384
vwsprintf Subroutine	384
wait, waitpid, wait3, or wait364 Subroutine	385
wscat, wcschr, wcsncmp, wcsncpy, or wcsncpy Subroutine	388
wscoll Subroutine	390
wcsftime Subroutine	391
wcsid Subroutine	392
wcslen Subroutine	393
wcsncat, wcsncmp, or wcsncpy Subroutine	394
wcsprbr Subroutine	395
wcsrchr Subroutine	396
wcsrtombs Subroutine	397
wcsspn Subroutine	398
wcsstr Subroutine	399
wcstod Subroutine	399
wcstok Subroutine	401
wcstol or wcstoll Subroutine	403
wcstombs Subroutine	405
wcstoul or wcstoull Subroutine	406
wcswcs Subroutine	408
wcswidth Subroutine	409
wcsxfrm Subroutine	410
wctob Subroutine	411
wctomb Subroutine	412
wctrans Subroutine	413
wctype or get_wctype Subroutine	413
wcwidth Subroutine	415
wlm_assign Subroutine	416
wlm_change_class Subroutine	418
wlm_check subroutine	420
wlm_classify Subroutine	421

wlm_class2key Subroutine.	423
wlm_create_class Subroutine	423
wlm_delete_class Subroutine.	425
wlm_endkey Subroutine	427
wlm_get_bio_stats subroutine	427
wlm_get_info Subroutine	430
wlm_init_class_definition Subroutine	432
wlm_initialize Subroutine	433
wlm_initkey Subroutine	434
wlm_key2class Subroutine.	435
wlm_load Subroutine.	436
wlm_read_classes Subroutine	438
wlm_set Subroutine	440
wlm_set_tag Subroutine	441
wmemchr Subroutine.	442
wmemcmp Subroutine	443
wmemcpy Subroutine	444
wmemmove Subroutine.	444
wmemset Subroutine.	445
wordexp Subroutine	446
wordfree Subroutine	448
write, writex, writev, writevx or pwrite Subroutines	449
wstring Subroutine	454
wstrtod or watof Subroutine	457
wstrtol, watol, or watoi Subroutine	458
yield Subroutine	459
Chapter 2. Curses Subroutines	461
addch, mvaddch, mvwaddch, or waddch Subroutine	461
addnstr, addstr, mvaddnstr, mvaddstr, mvwaddnstr, mvwaddstr, waddnstr, or waddstr Subroutine	462
attroff, attron, attrset, wattroff, wattron, or wattrset Subroutine.	464
attron or wattron Subroutine	466
attrset or wattrset Subroutine.	467
baudrate Subroutine	468
beep Subroutine	468
box Subroutine	469
can_change_color, color_content, has_colors,init_color, init_pair, start_color or pair_content Subroutine.	470
cbreak, nocbreak, noraw, or raw Subroutine	473
clear, erase, wclear or werase Subroutine	475
clearok, idlok, leaveok, scrollok, setscreg or wsetscreg Subroutine	476
clrtoebot or wclrtoebot Subroutine	479
clrtoeol or wclrtoeol Subroutine	480
color_content Subroutine	481
copywin Subroutine	482
curs_set Subroutine	483
def_prog_mode, def_shell_mode, reset_prog_mode or reset_shell_mode Subroutine	484
def_shell_mode Subroutine	485
del_curterm, restartterm, set_curterm, or setupterm Subroutine	486
delay_output Subroutine	488
delch, mvdelch, mvwdelch or wdelch Subroutine	489
deleteln or wdeleteln Subroutine	490
delwin Subroutine	491
echo or noecho Subroutine	492
echochar or wechochar Subroutines	493
endwin Subroutine	494

erase or werase Subroutine	494
erasechar, erasewchar, killchar, and killwchar Subroutine	495
filter Subroutine.	496
flash Subroutine	497
flushinp Subroutine	498
garbageclines Subroutine	499
getbegyx, getmaxyx, getparyx, or getyx Subroutine	500
getch, mvgetch, mvwgetch, or wgetch Subroutine	501
getmaxyx Subroutine.	505
getnstr, getstr, mvgetnstr, mvgetstr, mvwgetnstr, mvwgetstr, wgetnstr, or wgetstr Subroutine	506
getsyx Subroutine	508
getyx Macro	509
halfdelay Subroutine	510
has_colors Subroutine	511
has_ic and has_il Subroutine.	512
has_il Subroutine	513
idlok Subroutine	513
inch, mvinch, mvwinch, or winch Subroutine	514
init_color Subroutine	516
init_pair Subroutine	517
initscr and newterm Subroutine	518
insch, mvinsch, mvwinsch, or winsch Subroutine	519
insertln or winsertln Subroutine	520
intrflush Subroutine	521
keyname, key_name Subroutine	522
keypad Subroutine	523
killchar or killwchar Subroutine	524
_lazySetErrorHandler Subroutine	525
leaveok Subroutine	526
longname Subroutine	528
makenew Subroutine.	528
meta Subroutine	529
move or wmove Subroutine	530
mvcur Subroutine	531
mvwin Subroutine	532
newpad, pnoutrefresh, prefresh, or subpad Subroutine	533
newterm Subroutine	536
derwin, newwin, or subwin Subroutine	537
nl or nonl Subroutine.	540
nodelay Subroutine	540
notimeout, timeout, wtimeout Subroutine	541
overlay or overwrite Subroutine	543
pair_content Subroutine.	544
prefresh or pnoutrefresh Subroutine	545
printw, wprintw, mvprintw, or mvwprintw Subroutine	546
putp, tputs Subroutine	548
raw or noraw Subroutine	549
refresh or wrefresh Subroutine	550
reset_prog_mode Subroutine.	551
reset_shell_mode Subroutine.	552
resetterm Subroutine.	553
resetty, savetty Subroutine.	553
restartterm Subroutine	554
ripline Subroutine	555
savetty Subroutine	556
scanw, wscanw, mvscanw, or mvwscanw Subroutine	557

scr_dump, scr_init, scr_restore, scr_set Subroutine	558
scr_init Subroutine	560
scr_restore Subroutine	561
scrll, scroll, wscrll Subroutine	562
scrollok Subroutine	563
set_curterm Subroutine	564
setscrreg or wsetscrreg Subroutine	565
setsyx Subroutine	566
set_term Subroutine	567
setupterm Subroutine	568
_showstring Subroutine	569
slk_atroff, 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	570
slk_init Subroutine	573
slk_label Subroutine	574
slk_noutrefresh Subroutine	575
slk_refresh Subroutine	576
slk_restore Subroutine	577
slk_set Subroutine	577
slk_touch Subroutine	578
standend, standout, wstandend, or wstandout Subroutine	579
start_color Subroutine	580
subpad Subroutine	581
subwin Subroutine	582
tgetent, tgetflag, tgetnum, tgetstr, or tgoto Subroutine	584
tgetflag Subroutine	585
tgetnum Subroutine	586
tgetstr Subroutine	587
tgoto Subroutine	588
tigetflag, tigetnum, tigetstr, or tparm Subroutine	588
tigetnum Subroutine	590
tigetstr Routine	591
is_linetouched, is_wintouched, touchline, touchwin, untouchwin, or wtouchin Subroutine	592
touchoverlap Subroutine	594
touchwin Subroutine	595
tparm Subroutine	596
tputs Subroutine	597
typeahead Subroutine	598
unctrl Subroutine	599
ungetch, unget_wch Subroutine	600
vidattr, vid_attr, vidputs, or vid_puts Subroutine	601
doupdate, refresh, wnoutrefresh, or wrefresh Subroutines	602
Chapter 3. FORTRAN Basic Linear Algebra Subroutines (BLAS)	605
SDOT or DDOT Function	605
CDOTC or ZDOTC Function	605
CDOTU or ZDOTU Function	606
SAXPY, DAXPY, CAXPY, or ZAXPY Subroutine	607
SROTG, DROTG, CROTG, or ZROTG Subroutine	607
SROT, DROT, CSROT, or ZDROT Subroutine	608
SCOPY, DCOPY, CCOPY, or ZCOPY Subroutine	609
SSWAP, DSWAP, CSWAP, or ZSWAP Subroutine	610
SNRM2, DNRM2, SCNRM2, or DZNRM2 Function	611
SASUM, DASUM, SCASUM, or DZASUM Function	612
SSCAL, DSCAL, CSSCAL, CSCAL, ZDSCAL, or ZSCAL Subroutine	612
ISAMAX, IDAMAX, ICAMAX, or IZAMAX Function	613

SDSDOT Function	614
SROTM or DROTM Subroutine	615
SROTMG or DROTMG Subroutine	616
SGEMV, DGEMV, CGEMV, or ZGEMV Subroutine	617
SGBMV, DGBMV, CGBMV, or ZGBMV Subroutine	618
CHEMV or ZHEMV Subroutine	620
CHBMV or ZHBMV Subroutine	621
CHPMV or ZHPMV Subroutine	623
SSYMV or DSYMV Subroutine	624
SSBMV or DSBMV Subroutine	625
SSPMV or DSPMV Subroutine	626
STRMV, DTRMV, CTRMV, or ZTRMV Subroutine	627
STBMV, DTBMV, CTBMV, or ZTBMV Subroutine	629
STPMV, DTPMV, CTPMV, or ZTPMV Subroutine	631
STRSV, DTRSV, CTRSV, or ZTRSV Subroutine	632
STBSV, DTBSV, CTBSV, or ZTBSV Subroutine	634
STPSV, DTPSV, CTPSV, or ZTPSV Subroutine	636
SGER or DGER Subroutine	638
CGERU or ZGERU Subroutine	639
CGERC or ZGERC Subroutine	639
CHER or ZHER Subroutine	640
CHPR or ZHPR Subroutine	641
CHER2 or ZHER2 Subroutine	642
CHPR2 or ZHPR2 Subroutine	643
SSYR or DSYR Subroutine	644
SSPR or DSPR Subroutine	645
SSYR2 or DSYR2 Subroutine	646
SSPR2 or DSPR2 Subroutine	648
SGEMM, DGEMM, CGEMM, or ZGEMM Subroutine	649
SSYMM, DSYMM, CSYMM, or ZSYMM Subroutine	650
CHEMM or ZHEMM Subroutine	652
SSYRK, DSYRK, CSYRK, or ZSYRK Subroutine	654
CHERK or ZHERK Subroutine	655
SSYR2K, DSYR2K, CSYR2K, or ZSYR2K Subroutine	657
CHER2K or ZHER2K Subroutine	658
STRMM, DTRMM, CTRMM, or ZTRMM Subroutine	660
STRSM, DTRSM, CTRSM, or ZTRSM Subroutine	662
Appendix A. Base Operating System Error Codes for Services That Require Path-Name Resolution	665
Appendix B. ODM Error Codes	667
Appendix C. Notices	669
Index	673

About This Book

This book provides information on application programming interfaces to the operating system.

This book is part of the six-volume technical reference set, *AIX 5L Version 5.1 Technical Reference*, that provides information on system calls, kernel extension calls, and subroutines in the following volumes:

- *AIX 5L Version 5.1 Technical Reference: Base Operating System and Extensions Volume 1* and *AIX 5L Version 5.1 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 5.1 Technical Reference: Communications Volume 1* and *AIX 5L Version 5.1 Technical Reference: Communications Volume 2* provide information on entry points, functions, system calls, subroutines, and operations related to communications services.
- *AIX 5L Version 5.1 Technical Reference: Kernel and Subsystems Volume 1* and *AIX 5L Version 5.1 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.

Who Should Use This Book

This book is intended for experienced C programmers. To use the book effectively, you should be familiar with commands, system calls, subroutines, file formats, and special files.

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.
<i>Italics</i>	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.

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 UNIX98 Specification

Beginning with Version 4.3, the operating system is designed to support The Open Group's UNIX98 Specification 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 4.3 even more open and portable for applications.

At the same time, compatibility with previous releases of the operating system is preserved. This is accomplished by the creation of a new environment variable, which can be used to set the system environment on a per-system, per-user, or per-process basis.

To determine the proper way to develop a UNIX98-portable application, you may need to refer to The Open Group's UNIX98 Specification, which can be obtained on a CD-ROM by ordering *Go Solo 2: The Authorized Guide to Version 2 of the Single UNIX Specification*, ISBN: 0-13-575689-8, a book which includes The Open Group's UNIX98 Specification on a CD-ROM.

Related Publications

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

- *AIX 5L Version 5.1 System Management Guide: Operating System and Devices* ("About This Book" on page xiii)
 - *AIX 5L Version 5.1 System Management Guide: Communications and Networks*
 - *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*
 - *AIX 5L Version 5.1 Communications Programming Concepts*
 - *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts* ("About This Book" on page xiii)
 - *AIX 5L Version 5.1 Files Reference* ("About This Book" on page xiii)
-

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

- AIX
- AIXwindows

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be the trademarks or service marks of others.

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)
void * Base;
size_t NumberOfElements, Size;
int (*ComparisonPointer)(const void*, const void*);
```

Description

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

Parameters

<i>Base</i>	Points to the element at the base of the table.
<i>NumberOfElements</i>	Specifies the number of elements in the table.
<i>Size</i>	Specifies the size of each element.
<i>ComparisonPointer</i>	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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **bsearch** subroutine, **lsearch** subroutine.

Searching and Sorting Example Program, Subroutines Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

quotactl Subroutine

Purpose

Manipulates disk quotas.

Library

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

Syntax

```
#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.

Currently, disk quotas are supported only by the Journaled File System (JFS).

Parameters

Path Specifies the path name of any file within the mounted file system to which the quota control command is to be applied.

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

For JFSs, the *Cmd* parameter can be constructed through use of the **QCMD**(*Cmd*, *type*) macro contained within the **jfs/quota.h** file. The *Cmd* parameter specifies the quota control command. The *type* parameter specifies either user (**USRQUOTA**) or group (**GRPQUOTA**) quota type.

The valid JFS specific quota control values for the *Cmd* parameter 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.

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.

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.

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 **MINDQUID** through **MAXDQID**.

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:

EACCES	In the Q_QUOTAON command, the quota file is not a regular file.
EACCES	Search permission is denied for a component of a path prefix.
EFAULT	An invalid <i>Addr</i> parameter is supplied; the associated structure could not be copied in or out of the kernel.
EFAULT	The <i>Path</i> parameter points outside the process's allocated address space.
EINVAL	The specified quota control command or quota type is invalid.
EINVAL	Path name contains a character with the high-order bit set.
EINVAL	The <i>ID</i> parameter is outside of the supported range (MINDQUID through MAXDQID).
EIO	An I/O error occurred while reading from or writing to a file containing quotas.
ELOOP	Too many symbolic links were encountered in translating a path name.
ENAMETOOLONG	A component of either path name exceeded 255 characters, or the entire length of either path name exceeded 1023 characters.

ENOENT	A file name does not exist.
ENOTBLK	Mounted file system is not a block device.
ENOTDIR	A component of a path prefix is not a directory.
EOPNOTSUPP	The file system does not support quotas.
EPERM	The quota control commands is privileged and the caller did not have root user authority.
EROFS	In the Q_QUOTAON command, the quota file resides on a read-only file system.
EUSERS	The in-core quota table cannot be expanded.

Related Information

The **quotacheck** command.

Disk Quota System Overview in *AIX 5L Version 5.1 System Management Concepts: Operating System and Devices*.

How to Set Up the Disk Quota System in *AIX 5L Version 5.1 System Management Guide: Operating System and Devices*.

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);
```

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

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

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

Related Information

The **_exit** subroutine, **kill** subroutine, **pthread_kill** subroutine, **sigaction** (“sigaction, sigvec, or signal Subroutine” on page 135) subroutine.

Signal Management in *AIX 5L Version 5.1 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

```
#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** (“rand_r Subroutine” on page 6) 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:

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

void srand (Seed)
```

```
unsigned
int Seed;
{
    next = Seed;
}
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **drand48**, **erand48**, **lrand48**, **nrand48**, **mrnd48**, **jrnd48**, **srand48**, **seed48**, or **lcng48** subroutine, **random**, **srandom**, **initstate**, or **setstate** (“random, srandom, initstate, or setstate Subroutine” on page 8) subroutine.

Subroutines Overview in *AIX 5L Version 5.1 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** (“random, srandom, initstate, or setstate Subroutine” on page 8) subroutine for more elaborate random-number generators that have greater spectral properties.

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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

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

File

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

Related Information

The **drand48** subroutine, **random** (“random, srandom, initstate, or setstate Subroutine” on page 8) subroutine.

random, srand, initstate, or setstate Subroutine

Purpose

Generates pseudo-random numbers more efficiently.

Library

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

Syntax

```
#include <stdlib.h>
long random ( )
void srand (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**, **srand**, **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 $16 * (2^{31}-1)$. 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 **random** 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 **srand** 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 **random** 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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **drand48**, **erand48**, **jrand48**, **lcong48**, **lrand48**, **mrand48**, **nrand48**, **seed48**, or **srand48** subroutine, **rand** or **srand** (“rand or srand Subroutine” on page 5) subroutine.

Subroutines Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

read, readx, readv, readvx, or pread Subroutine

Purpose

Reads from a file.

Library

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

Syntax

```
#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;

int 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.

Note: A read from a regular file is not interruptible. Only reads from objects that may block indefinitely, such as FIFOs, sockets, and some devices, are generally interruptible.

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.

```
ssize_t pread64(int filedes , 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.

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** (“ulimit Subroutine” on page 362) 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 **adirent** 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:

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

<i>iovCount</i>	Specifies the number of iovec structures pointed to by the <i>iov</i> parameter.
<i>Buffer</i>	Points to the buffer.
<i>NBytes</i>	Specifies the number of bytes read from the file associated with the <i>FileDescriptor</i> 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**, **readv**, **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**, **readv**, **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 <i>FileDescriptor</i> parameter is not a valid file descriptor open for reading.
EINVAL	The file position pointer associated with the <i>FileDescriptor</i> parameter was negative.
EINVAL	The sum of the iov_len values in the <i>iov</i> array was negative or overflowed a 32-bit integer.
EINVAL	The value of the <i>iovCount</i> parameter was not between 1 and 16, inclusive.
EINVAL	The STREAM or multiplexer referenced by <i>FileDescriptor</i> 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 <i>Buffer</i> or part of the <i>iov</i> 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.

Note: The **E_OVERFLOW** error code applies to AIX 4.2 and later releases.

E_OVERFLOW	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 <i>FileDescriptor</i> .
-------------------	--

The **read**, **readx**, **readv**, **readvx** 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	<i>fildev</i> 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.
------------------	---------------------------

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

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 Input and Output Handling in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

readdir_r Subroutine

Purpose

Reads a directory.

Library

Thread-Safe C Library (**libc_r.a**)

Syntax

```
#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 a 0. 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.

Parameters

<i>DirectoryPointer</i>	Points to the DIR structure of an open directory.
<i>Entry</i>	Points to a structure that contains the next directory entry.
<i>Result</i>	Points to the directory entry specified by the <i>Entry</i> parameter.

Return Values

0 Indicates that the subroutines was successful.
9 Indicates that the subroutines was not successful.

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 <i>DirectoryPointer</i> parameter, or read permission is denied for the structure pointed to by the <i>DirectoryPointer</i> parameter.
---------------	---

ENAMETOOLONG	The length of the <i>DirectoryPointer</i> 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 <i>DirectoryPointer</i> 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 <i>DirectoryPointer</i> parameter does not refer to an open directory stream.

Examples

To search a directory for the entry name, enter:

```
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;
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

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

Related Information

The **close** subroutine, **exec** subroutines, **fork** subroutine, **lseek** subroutine, **openx**, **open**, or **creat** subroutine, **read**, **readv**, **readx**, or **readvx** (“read, readx, readv, readvx, or pread Subroutine” on page 9) subroutine, **scandir** or **alphasort** (“scandir or alphasort Subroutine” on page 80) subroutine.

The **opendir**, **readdir**, **telldir**, **seekdir**, **rewinddir**, or **closedir** subroutine.

Subroutines Overview, List of File and Directory Manipulation Services, and List of Multithread Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

readlink Subroutine

Purpose

Reads the contents of a symbolic link.

Library

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

Syntax

```
#include <unistd.h>
int readlink ( Path, Buffer, BufferSize)
```

```
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

<i>Path</i>	Specifies the path name of the destination file or directory.
<i>Buffer</i>	Points to the user's buffer. The buffer should be at least as large as the <i>BufferSize</i> parameter.
<i>BufferSize</i>	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:

ENOENT	The file named by the <i>Path</i> parameter does not exist, or the path points to an empty string.
EINVAL	The file named by the <i>Path</i> parameter is not a symbolic link.
ERANGE	The path name in the symbolic link is longer than the <i>BufferSize</i> value.

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.
------------------	---------------------------

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **In** command.

The **link** subroutine, **statx**, **stat**, **fstatx**, **fstat**, **fullstat**, or **ffullstat** (“statx, stat, lstat, fstatx, fstat, fullstat, ffullstat, stat64, lstat64, or fstat64 Subroutine” on page 251) subroutine, **symlink** (“symlink Subroutine” on page 278) subroutine, **unlink** (“unlink Subroutine” on page 369) subroutine.

Files, Directories, and File Systems for Programmers in *AIX 5L Version 5.1 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

```
#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) 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 **RTC_POWER_PC**.

The **time_base_to_time** subroutine converts time base information to real time, if necessary. It is recommended that applications unconditionally call the **time_base_to_time** subroutine rather than performing a check to see if it is necessary.

If *t->flag* is **RTC_POWER**, the subroutine simply returns (the data is already in real time format).

If *t->flag* is **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 **RTC_POWER**.

Parameters

t Points to a *timebasestruct_t*.

Return Values

The **read_real_time** subroutine returns **RTC_POWER** if the contents of the real time clock has been recorded in the *timebasestruct*, or returns **RTC_POWER_PC** if the content of the time base registers has been recorded in the *timebasestruct*.

The **time_base_to_time** subroutine returns **0** if the conversion to real time is successful (or not necessary), otherwise **-1** is returned.

Examples

This example shows the time it takes for **printf** to print the comment between the begin and end time codes:

```
#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);
}

```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **gettimer**, **settimer**, **restimer**, **stime**, or **time** subroutines, **getrusage**, **times**, or **vtimes** subroutines.

High-Resolution Time Measurements Using POWER-based Time Base or POWER family Real-Time Clock in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

realpath Subroutine

Purpose

Resolves path names.

Library

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

Syntax

#include <stdlib.h>

```

char *realpath (const char *file_name,
char *resolved_name)

```

Description

The **realpath** subroutine performs filename expansion and path name resolution in *file_name* and stores it in *resolved_name*.

The **realpath** subroutine can handle both relative and absolute path names. For both absolute and relative path names, the **realpath** subroutine returns the resolved absolute path name.

The character pointed to by *resolved_name* must be big enough to contain the fully resolved path name. The value of **PATH_MAX** (defined in **limits.h** header file) may be used as an appropriate array size.

Return Values

On successful completion, the **realpath** subroutine returns a pointer to the resolved name. Otherwise, it returns a null pointer, and sets **errno** to indicate the error. If the **realpath** subroutine encounters an error, the contents of *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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **getcwd** or **sysconf** (“sysconf Subroutine” on page 282) subroutine.

reboot Subroutine

Purpose

Restarts the system.

Library

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

Syntax

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

```
void reboot ( HowTo, 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

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

```
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 \)

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

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 in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

National Language Support Overview for Programming and Subroutines Overview in *AIX 5L Version 5.1 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

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.

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:

```
/* . . . Your Program . . . */
malloc(n)
    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 . . . xXyYzZ], 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.

Matching a Character Class in National Language Support

A common use of the range expression is matching a character class. For example, [0-9] represents all digits, and [a-z, A-Z] represents all letters. This form may produce unexpected results when ranges are interpreted according to the current collating sequence.

Instead of the range expression shown above, use a character class expression within brackets to match characters. The system interprets this type of expression according to the current character class definition. However, you cannot use character class expressions in range expressions.

The following exemplifies the syntax of a character class expression:

```
[:charclass:]
```

that is, a left bracket followed by a colon, followed by the name of the character class, followed by another colon and a right bracket.

National Language Support supports the following character classes:

[upper:]	ASCII uppercase letters.
[lower:]	ASCII lowercase letters.
[alpha:]	ASCII uppercase and lowercase letters.
[digit:]	ASCII digits.
[alnum:]	ASCII uppercase and lowercase letters, and digits.
[xdigit:]	ASCII hexadecimal digits.
[punct:]	ASCII punctuation character (neither a control character nor an alphanumeric character).
[space:]	ASCII space, tab, carriage return, new-line, vertical tab, or form feed character.
[print:]	ASCII printing characters.

Parameters

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

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **ctype** subroutine, **compile**, **step**, or **advance** subroutine, **malloc**, **free**, **realloc**, **calloc**, **mallopt**, **mallinfo**, or **alloca** subroutine, **regcomp** (“regcomp Subroutine”) subroutine, **regex** (“regex Subroutine” on page 29) subroutine.

The **ed** command, **regcmp** command.

Subroutines Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

regcomp Subroutine

Purpose

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

Library

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

Syntax

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

```
int regcomp ( Preg, Pattern, CFlags)  
const char *Preg;  
const char *Pattern;  
int 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

<i>Preg</i>	Specifies the structure to receive the compiled output of the regcomp subroutine.
<i>Pattern</i>	Contains the basic or extended regular expression to be compiled by the regcomp subroutine. The default regular expression type for the <i>Pattern</i> parameter is a basic regular expression. An application can specify extended regular expressions with the REG_EXTENDED flag.
<i>CFlags</i>	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 regex subroutine. If this flag is not set, the regcomp subroutine sets the re_nsub structure to the number of parenthetical expressions found in the <i>Pattern</i> 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):

```
#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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **regerror** (“regerror Subroutine”) subroutine, **regexec** (“regexec Subroutine” on page 29) subroutine, **regfree** (“regfree Subroutine” on page 32) subroutine.

Subroutines Overview and Understanding Internationalized Regular Expression Subroutines in *AIX 5L Version 5.1 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

```
#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 **regex** 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 **regex** 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

<i>ErrCode</i>	Specifies the error for which a description string is to be returned.
<i>Preg</i>	Specifies the structure that holds the previously compiled output of the regcomp subroutine.
<i>ErrBuf</i>	Specifies the buffer to receive the string generated by the regerror subroutine.
<i>ErrBuf_Size</i>	Specifies the size of the <i>ErrBuf</i> 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_ETYPE	Indicates a character class-type reference that is not valid.
REG_ESCAPE	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_NEWLINE	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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **regcomp** (“regcomp Subroutine” on page 25) subroutine, **regexec** (“regexec Subroutine”) subroutine, **regfree** (“regfree Subroutine” on page 32) subroutine.

Subroutines Overview and Understanding Internationalized Regular Expression Subroutines in *AIX 5L Version 5.1 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 parenthetical 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 parenthetical 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 **`regexexec`** 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 $\hat{\text{}}$ (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 $[\hat{\text{}}abc]$ 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 $\hat{\text{}}$ (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

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

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 $\hat{\ } (circumflex)$, 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 **regexec** 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 **regexec** 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_ETYPE	Indicates a character class-type reference that is not valid.
REG_EESCAPE	Indicates a trailing $\backslash (backslash)$ in the pattern.
REG_ESUBREG	Indicates a number in $\backslash digit$ is not valid or is in error.
REG_EBRACK	Indicates a $[] (left\ and\ right\ brackets)$ imbalance.
REG_EPAREN	Indicates a $\backslash (\backslash) (backslash, left\ parenthesis, backslash, right\ parenthesis)$ or $() (left\ and\ right\ parentheses)$ imbalance.
REG_EBRACE	Indicates a $\backslash \{ \} (backslash, left\ brace, backslash, right\ brace)$ imbalance.
REG_BADBR	Indicates the content of $\backslash \{ \} (backslash, left\ brace, backslash, right\ brace)$ 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.)

```
(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) ;
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **regcomp** (“regcomp Subroutine” on page 25) subroutine, **regerror** (“regerror Subroutine” on page 27) subroutine, **regfree** (“regfree Subroutine”) subroutine.

Subroutines Overview and Understanding Internationalized Regular Expression Subroutines in *AIX 5L Version 5.1 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

```
#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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **regcomp** (“regcomp Subroutine” on page 25) subroutine, **regerror** (“regerror Subroutine” on page 27) subroutine, **regexec** (“regexec Subroutine” on page 29) subroutine.

Subroutines Overview and Understanding Internationalized Regular Expression Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

retimerid Subroutine

Purpose

Releases a previously allocated interval timer.

Library

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

Syntax

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

```
int retimerid ( TimerID)
timer_t TimerID;
```

Description

The **retimerid** 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 **retimerid** 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 **retimerid** 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **gettimerid** subroutine.

List of Time Data Manipulation Services in *AIX 5L Version 5.1 System Management Concepts: Operating System and Devices*.

remove Subroutine

Purpose

Removes a file.

Library

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

Syntax

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

```
int remove( FileName)  
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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **link** subroutine, **rename** (“rename Subroutine”) subroutine.

The **link** or **unlink** (“unlink Subroutine” on page 369) command.

Files, Directories, and File Systems for Programmers in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

rename Subroutine

Purpose

Renames a directory or a file.

Library

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

Syntax

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

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

<i>FromPath</i>	Identifies the file or directory to be renamed.
<i>ToPath</i>	Identifies the new path name of the file or directory to be renamed. If <i>ToPath</i> is an existing file or empty directory, it is replaced by <i>FromPath</i> . If <i>ToPath</i> 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 <i>FromPath</i> or <i>ToPath</i> parameter is currently in use by the system, or the file named by <i>FromPath</i> or <i>ToPath</i> is a named STREAM.
EDQUOT	The directory that would contain the path specified by the <i>ToPath</i> 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 <i>ToPath</i> parameter specifies an existing directory that is not empty.
EINVAL	The path specified in the <i>FromPath</i> or <i>ToPath</i> parameter is not a well-formed directory (<i>FromPath</i> is an ancestor of <i>ToPath</i>), or an attempt has been made to rename . (dot) or .. (dot dot).
EISDIR	The <i>ToPath</i> parameter names a directory and the <i>FromPath</i> parameter names a non-directory.
EMLINK	The <i>FromPath</i> parameter names a directory that is larger than the maximum link count of the parent directory of the <i>ToPath</i> parameter.
ENOENT	A component of either path does not exist, the file named by the <i>FromPath</i> parameter does not exist, or a symbolic link was named, but the file to which it refers does not exist.
ENOSPC	The directory that would contain the path specified in the <i>ToPath</i> parameter cannot be extended because the file system is out of space.
ENOTDIR	The <i>FromPath</i> parameter names a directory and the <i>ToPath</i> parameter names a non-directory.
ENOTEMPTY	The <i>ToPath</i> parameter specifies an existing directory that is not empty.
EROFS	The requested operation requires writing in a directory on a read-only file system.
ETXTBSY	The <i>ToPath</i> parameter names a shared text file that is currently being used.
EXDEV	The link named by the <i>ToPath</i> parameter and the file named by the <i>FromPath</i> parameter are on different file systems.

If Network File System (NFS) is installed on the system, 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **chmod** subroutine, **link** subroutine, **mkdir** subroutine, **rmdir** ("rmdir Subroutine" on page 38) subroutine, **unlink** ("unlink Subroutine" on page 369) subroutine.

The **chmod** command, **mkdir** command, **mv** command, **mvd** command.

revoke Subroutine

Purpose

Revokes access to a file.

Library

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

Syntax

```
int revoke ( Path )
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 <i>Path</i> 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **chmod** subroutine, **frevoke** subroutine.

List of Security and Auditing Subroutines and Subroutines Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

rmdir Subroutine

Purpose

Removes a directory.

Library

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

Syntax

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

```
int rmdir ( Path)
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:

EACCES	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 <i>Path</i> parameter is not empty.
ENAMETOOLONG	The length of the <i>Path</i> 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 <i>Path</i> parameter does not exist, or the <i>Path</i> parameter points to an empty string.
ENOTDIR	A component specified by the <i>Path</i> parameter is not a directory.
EINVAL	The directory named by the <i>Path</i> parameter is not well-formed.
EROFS	The directory named by the <i>Path</i> 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **chmod** or **fchmod** subroutine, **mkdir** subroutine, **remove** ("remove Subroutine" on page 34) subroutine, **rename** ("rename Subroutine" on page 34) subroutine, **umask** ("umask Subroutine" on page 364) subroutine, **unlink** ("unlink Subroutine" on page 369) subroutine.

The **rm** command, **rmdir** command.

Files, Directories, and File Systems For Programmers in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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 ( Response )
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][:a1pha:]* | ok
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **localeconv** subroutine, **nl_langinfo** subroutine, **regcomp** (“regcomp Subroutine” on page 25) subroutine, **regexec** (“regexec Subroutine” on page 29) subroutine, **setlocale** (“setlocale Subroutine” on page 107) subroutine.

National Language Support Overview for Programming and Understanding Locale Subroutines in *AIX 5L Version 5.1 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**)

Syntax

```
#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 excp_type;
int severity;
int trap_no;
```

Parameters

rhandle Must be an **RSiHandle**, which was previously initialized by the **RSiOpen** (“**RSiOpen Subroutine**” on page 67) subroutine.

HotSet Specifies a pointer to a valid structure of type **SpmiHotSet** as created by the **RSiCreateHotSet** (“**RSiCreateHotSet Subroutine**” on page 46) 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 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 **RSiGetHotItem** (“**RSiGetHotItem Subroutine**” on page 54) 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/armpeek**) is still valid and won't involve creation of pseudo **SpmiHotVals** structures.

maxresp Must be non-zero if *excp_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 $2 \times 3 = 6$ 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 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

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* .

Implementation Specifics

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

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 46
- “RSiOpen Subroutine” on page 67.

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, msec)
RSiHandle rhandle; struct SpmiStatSet *statset; int msec;
```

Parameters

rhandle Must be an **RSiHandle**, which was previously initialized by the **RSiOpen** (“**RSiOpen Subroutine**” on page 67) 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 71) 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**.

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*.

Implementation Specifics

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

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 47
- “RSiOpen Subroutine” on page 67
- “RSiStartFeed Subroutine” on page 71.

RSiChangeHotFeed 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

```
#include sys/Rsi.h
int RSiChangeFeed(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 67) 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 46) subroutine call. Data feeding must have been started for this **SpmiHotSet** via a previous **RSiStartHotFeed** (“**RSiStartHotFeed Subroutine**” on page 72) 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**.

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*.

Implementation Specifics

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

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

In the sample program, the **SpmiStatSet** is created in the local function **lststats** shown previously in lines 6 through 10.

- “**RsiCreateHotSet Subroutine**” on page 46
- “**RSiOpen Subroutine**” on page 67
- “**RSiStartHotFeed Subroutine**” on page 72.

RSiClose Subroutine

Purpose

Terminates the RSI interface for a remote host connection.

Library

RSI Library (**libSpmi.a**)

Syntax

```
#include sys/Rsi.h
void RSiClose(rhandle)
RSiHandle rhandle;
```

Description

The **RSiClose** subroutine is responsible for:

1. Removing the data-consumer program as a known data consumer on a particular host. This is done by sending a **going_down** packet to the host.
2. Marking the RSI handle as not active.
3. Releasing all memory allocated in connection with the RSI handle.
4. Terminating the RSI interface for a remote host.

A successful **RSiOpen** (“**RSiOpen Subroutine**” on page 67) subroutine creates tables on the remote host it was issued against. Therefore, a data consumer program that has issued successful **RSiOpen** subroutine calls should issue an **RSiClose** (“**RSiClose Subroutine**” on page 45) subroutine call for each **RSiOpen** call before the program exits so that the tables in the remote **xmservd** daemon can be released.

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).

Implementation Specifics

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

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:

- “RSiInit Subroutine” on page 58
- “RSiOpen Subroutine” on page 67

RSiCreateHotSet Subroutine

Purpose

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

Library

RSI Library (**libSpmi.a**)

Syntax

```
#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** (“**RSiAddSetHot Subroutine**” on page 41) subroutine can be successfully called.

Parameters

rhandle Must be an **RSiHandle**, which was previously initialized by the **RSiOpen** (“**RSiOpen Subroutine**” on page 67) 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* .

Implementation Specifics

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

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 67
- “**RSiAddSetHot Subroutine**” on page 41.

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 69) subroutine can be successfully called.

Parameters

rhandle Must be an **RSiHandle**, which was previously initialized by the **RSiOpen** (“**RSiOpen Subroutine**” on page 67) 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*.

Implementation Specifics

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

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 67
- “RSiPathAddSetStat Subroutine” on page 69.

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.

Parameters

rhandle Must be an **RSiHandle**, which was previously initialized by the **RSiOpen** (“**RSiOpen Subroutine**” on page 67) 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** (“**RSiAddSetHot Subroutine**” on page 41) subroutine call. You cannot specify an **SpmiHotVals** that was internally generated by the Spmi library code as described under the *GrandParent* parameter to **RSiAddSetHot** (“**RSiAddSetHot Subroutine**” on page 41).

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*.

Implementation Specifics

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

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 67
- “**RSiAddSetHot Subroutine**” on page 41.

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.

Parameters

rhandle Must be an **RSiHandle**, which was previously initialized by the **RSiOpen** (“**RSiOpen Subroutine**” on page 67) subroutine.

ssp Must be a pointer to a structure type **struct SpmiStatSet**, which was previously returned by a successful **RSiCreateStatSet** (“**RSiCreateStatSet Subroutine**” on page 47) subroutine call.

svp Must be a handle of type **struct SpmiStatVals** as returned by a successful **RSiPathAddSetStat** (“**RSiPathAddSetStat Subroutine**” on page 69) 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*.

Implementation Specifics

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

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 47
- “RSiOpen Subroutine” on page 67
- “RSiPathAddSetStat Subroutine” on page 69.

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;
cx_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.

Parameters

rhandle Must be an **RSiHandle**, which was previously initialized by the **RSiOpen** (“**RSiOpen Subroutine**” on page 67) subroutine.

context Must be a handle of type **cx_handle**, which was previously returned by a successful **RSiPathGetCx** (“**RSiPathGetCx Subroutine**” on page 70) 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 *List of RSi Error Codes*.

Implementation Specifics

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

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 64
- “RSiOpen Subroutine” on page 67
- “RSiPathGetCx Subroutine” on page 70.

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)
RSiHandle rhandle;
cx_handle *context;
char **name;
char **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.

Parameters

rhandle Must be an **RSiHandle**, which was previously initialized by the **RSiOpen** (“**RSiOpen Subroutine**” on page 67) subroutine.

context Must be a handle of type **cx_handle**, which was previously returned by a successful **RSiPathGetCx** (“**RSiPathGetCx Subroutine**” on page 70) 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, 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*.

Implementation Specifics

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

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 65
- “RSiOpen Subroutine” on page 67
- “RSiPathGetCx Subroutine” on page 70.

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

```
#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 **RSiGetHotItem** 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*.

Parameters

rhandle	Must be an RSiHandle , which was previously initialized by the RSiOpen (“ RSiOpen Subroutine ” on page 67) subroutine.
HotSet	Used to return a pointer to a valid SpmiHotSet structure as obtained by a previous RSiCreateHotSet (“ RSiCreateHotSet Subroutine ” on page 46) subroutine call. The calling program can use this value to locate the SpmiHotSet if its address was stored by the program after it was created. The time stamps in the SpmiHotSet are updated with the time stamps of the decoded SpmiHotItems element.
index	A pointer to an integer that contains the desired relative element number in the SpmiHotItems array across all SpmiStatVals contained in the data packet. A value of zero points to the first element. When the RSiGetHotItem subroutine returns, the integer contains the index of the next SpmiHotItems element in the data packet. By passing the returned <i>index</i> parameter to the next call to RSiGetHotItem , the calling program can iterate through all SpmiHotItems elements in the hot_feed data packet.

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 RSiGetHotItem function: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – 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.
absvalue	A pointer to a float variable. A successful call will return the decoded value of the val field of the SpmiHotItems structure of the peer statistic. In case of a statistic of type SiQuantity , this value will be the same as the one returned in the argument <i>value</i> . In case of a peer statistic of type SiCounter , the value returned is the absolute value of the counter.
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 **RSiGetHotItem** subroutine returns a pointer to the current **SpmiHotVals** structure within the hotset. If no more **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 **SpmiGetHotSet** subroutine call to end-users. In the returned **SpmiHotVals** structure, all fields contain the correct values as declared, except for the following:

stat	Declared as SpmiStatHdl , actually points to a valid SpmiStat structure. By casting the handle to a pointer to SpmiStat , data in the structure can be accessed.
grandpa	Contains the cx_handle for the parent context of the peer contexts.
items	When using the Spmi interface this is an array of SpmiHotItems structures. When using the RSiGetHotItem subroutine, the array is empty and attempts to access it will likely result in segmentation faults or access of not valid data.
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 SpmiHotItems 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 SpmiHotItems 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 *List of RSi Error Codes*.

Implementation Specifics

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

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 67
- “RSiCreateHotSet Subroutine” on page 46.

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.

Parameters

rhandle Must be an **RSiHandle**, which was previously initialized by the **RSiOpen** (“**RSiOpen Subroutine**” on page 67) subroutine.

svp A handle of type **struct SpmiStatVals**, which was previously returned by a successful **RSiPathAddSetStat** (“**RSiPathAddSetStat Subroutine**” on page 69) 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* .

Implementation Specifics

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

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 67
- “RSiPathAddSetStat Subroutine” on page 69.

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.

- 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.

Parameters

rhandle Must be an **RSiHandle**, previously initialized by the **RSiOpen** (“**RSiOpen Subroutine**” on page 67) subroutine.

svp A handle of type **struct SpmiStatVals**, which was previously returned by a successful **RSiPathAddSetStat** (“**RSiPathAddSetStat Subroutine**” on page 69) 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 **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*.

Implementation Specifics

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

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 67
- “**RSiPathAddSetStat Subroutine**” on page 69

RSiInit Subroutine

Purpose

Allocates or changes the table of RSi handles.

Library

RSI Library (**libSpmi.a**)

Syntax

```
#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.

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:

- 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*.

Implementation Specifics

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

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 45.

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;
cx_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 **SiCflInst** 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.

Parameters

rhandle Must point to a structure of type **RSiHandle**, which was previously initialized by the **RSiOpen** (“**RSiOpen Subroutine**” on page 67) subroutine.

context Must be a handle of type **cx_handle**, which was previously returned by a successful **RSiPathGetCx** (“**RSiPathGetCx Subroutine**” on page 70) 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*.

Implementation Specifics

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

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 51
- “RSiOpen Subroutine” on page 67
- “RSiPathGetCx Subroutine” on page 70.

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.

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 67)** 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*.

Implementation Specifics

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

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 67.

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 67) call has been successfully completed and that the connection has not been closed.

Parameters

msecsThe 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*.

Implementation Specifics

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

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 67.

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.

Parameters

rhandle Must point to a structure of type **RSiHandle**, which was previously initialized by the **RSiOpen** (“**RSiOpen Subroutine**” on page 67) subroutine.

context Must be a handle of type **cx_handle**, which was previously returned by a successful **RSiPathGetCx** (“**RSiPathGetCx Subroutine**” on page 70) 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 51) 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 **RSiErrMsg**.

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*.

Implementation Specifics

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

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 51
- “RSiOpen Subroutine” on page 67
- “RSiPathGetCx Subroutine” on page 70.

RSiNextStat Subroutine

Purpose

Returns the next statistic of an **SpmiCx** context.

Library

RSI Library (**libSpmi.a**)

Syntax

```
#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.

Parameters

rhandle Must be an **RSiHandle**, which was previously initialized by the **RSiOpen** (“**RSiOpen Subroutine**” on page 67) subroutine.

context Must be a handle of type **cx_handle**, which was previously returned by a successful **RSiPathGetCx** (“**RSiPathGetCx Subroutine**” on page 70) subroutine call.

link Must be a pointer to a structure of type **struct SpmiStatLink**, which was previously returned by a successful **RSiFirstStat** (“**RSiFirstStat Subroutine**” on page 52) 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*.

Implementation Specifics

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

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 52
- “**RSiOpen Subroutine**” on page 67
- “**RSiPathGetCx Subroutine**” on page 70.

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.

Parameters

The arguments to the subroutine are:

rhandle Must point to an element of the **RSiHandleStruct** array, which is returned by a previous **RSiInit** (“**RSiInit Subroutine**” on page 58) call. 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 61) or **RSiOpen** (“**RSiOpen Subroutine**” on page 67) 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 45) 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 *List of RSi Error Codes*.

Implementation Specifics

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

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 45
- “RSIInvite Subroutine” on page 61

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 67) 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 47) 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*.

Implementation Specifics

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

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 47
- “RSiOpen Subroutine” on page 67.

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.

Parameters

rhandle Must be an **RSiHandle**, which was previously initialized by the **RSiOpen** (“**RSiOpen Subroutine**” on page 67) 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*.

Implementation Specifics

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

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 51
- “RSiOpen Subroutine” on page 67
- “RSiNextCx Subroutine” on page 64.

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.

Parameters

rhandle Must be an **RSiHandle**, which was previously initialized by the **RSiOpen** (“**RSiOpen Subroutine**” on page 67) 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 47) 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**.

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*.

Implementation Specifics

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

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 47
- “**RSiOpen Subroutine**” on page 67
- “**RSiStopFeed Subroutine**” on page 75.

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.

Parameters

rhandle Must be an **RSiHandle**, which was previously initialized by the **RSiOpen** (“**RSiOpen Subroutine**” on page 67) subroutine.

hotset Must be a pointer to a structure of type **struc SpmiHotSet**, which was previously returned by a successful **RSiCreateHot** (“**RSiCreateHotSet Subroutine**” on page 46) 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 **RSiErrMsg**.

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 **RSiErrorType**. RSi error codes are described in *List of RSi Error Codes*.

Implementation Specifics

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

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 46
- “RSiOpen Subroutine” on page 67
- “RSiChangeHotFeed Subroutine” on page 44
- “RSiStopHotFeed Subroutine” on page 76.

RSiStatGetPath Subroutine

Purpose

Finds the full path name of a statistic identified by a **SpmiStatVals** pointer.

Library

RSI Library (**libSpmi.a**)

Syntax

```
#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.

Parameters

rhandle Must be an **RSiHandle**, previously initialized by the **RSiOpen** (“**RSiOpen Subroutine**” on page 67) subroutine.

svp Must be a handle of type **struct SpmiStatVals** as returned by a successful **RSiPathAddSetStat** (“**RSiPathAddSetStat Subroutine**” on page 69) 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 **RSiErrMsg**.

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 **RSiErrorType**. RSI error codes are described in *List of RSI Error Codes*.

Implementation Specifics

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

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 67
- “RSiPathAddSetStat Subroutine” on page 69.

RSiStopFeed Subroutine

Purpose

Tells `xmservd` to stop sending data feeds for a `statset`.

Library

RSI Library (`libSpmi.a`)

Syntax

```
#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 (“RSiStartFeed Subroutine” on page 71)** 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.

Parameters

rhandle Must point to a structure of type **RSiHandle**, which was previously initialized by the **RSiOpen (“RSiOpen Subroutine” on page 67)** 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 47)** subroutine call. Data feeding must have been started for this **SpmiStatSet** via a previous **RSiStartFeed (“RSiStartFeed Subroutine” on page 71)** 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 **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 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*.

Implementation Specifics

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

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 67
- “RSiStartFeed Subroutine” on page 71.

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

```
#include sys/Rsi.h
int RSiStopFeed(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 72) 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.

Parameters

rhandle Must point to a structure of type **RSiHandle**, which was previously initialized by the **RSiOpen** (“**RSiOpen Subroutine**” on page 67) 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 46) subroutine call. Data feeding must have been started for this **SpmiStatSet** via a previous **RSiStartHotFeed** (“**RSiStartHotFeed Subroutine**” on page 72) subroutine call.

erase If 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*.

Implementation Specifics

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

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 67
- “**RSiStartHotFeed Subroutine**” on page 72
- “**RSiChangeHotFeed Subroutine**” on page 44.

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 ($1.0/\text{sqrt}(x)$). On some platforms, using the **rsqrt** subroutine is faster than computing $1.0 / \text{sqrt}(x)$. 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 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 If x is 1.0.
+0.0 If x is +INF.

Error Codes

When using either the **libm.a** or **libmsaa.a** library, the **rsqrt** subroutine may return the following error code:

EDOM The value of x is negative.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **matherr** subroutine, **sqrt** or **cbrt** (“sqrt, sqrtl, or cbrt Subroutine” on page 211) 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**.

Implementation Specifics

These subroutines are part of the Base Operating System (BOS) Runtime.

Files

`/usr/include/rpcsvc/rstat.x`

Related Information

The **rup** command.

The **rstatd** daemon

__safe_fetch Subroutine

Purpose

Reads the value of a single word variable protected by a lock.

Library

Standard C library (**libc.a**)

Syntax

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

```
int _safe_fetch ( word_addr)  
atomic_p word_addr;
```

Parameter

word_addr Specifies the address of the single word variable.

Description

The **_safe_fetch** subroutine safely reads and returns a single word value that is protected by a lock. This subroutine is used to read protected data before releasing the lock word with the **_clear_lock** subroutine. If **_safe_fetch** is not used, instructions that access data just before a lock release could actually be performed after the lock release.

Note: The word variable must be aligned on a full word boundary.

Return Values

This subroutine returns the value of the single word variable.

Implementation Specifics

The **_safe_fetch** subroutine is part of Base Operating System (BOS) Runtime.

Note that the **_safe_fetch** subroutine is intended for use only with the **_check_lock** and **_clear_lock** subroutines.

Related Information

The **_check_lock** subroutine, **_clear_lock** subroutine.

scandir or alphasort 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;
```


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.

Parameters

<i>DirectoryName</i>	Points to the directory name.
<i>NameList</i>	Points to the array of pointers to directory entries.
<i>Select</i>	Points to a user-supplied subroutine that is called by the scandir subroutine to select which entries to include in the array.
<i>Compare</i>	Points to a user-supplied subroutine that sorts the completed array.
<i>Directory1, Directory2</i>	Point to dirent structures.

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 <i>Directory1</i> parameter is lexically less than the dirent structure pointed to by the <i>Directory2</i> parameter.
0	The dirent structures pointed to by the <i>Directory1</i> parameter and the <i>Directory2</i> parameter are equal.
Greater than 0	The dirent structure pointed to by the <i>Directory1</i> parameter is lexically greater than the dirent structure pointed to by the <i>Directory2</i> parameter.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **malloc**, **free**, **realloc**, **calloc**, **mallopt**, **mallinfo**, or **alloca** subroutine, **opendir**, **readdir**, **telldir**, **seekdir**, **rewinddir**, or **closedir** subroutine, **qsort** (“qsort Subroutine” on page 1) subroutine.

Files, Directories, and File Systems for Programmers in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

scanf, fscanf, sscanf, or wscanf Subroutine

Purpose

Converts formatted input.

Library

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

or (**libc128.a**)

Syntax

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

```
int scanf ( Format [, Pointer, ... ] )  
const char *Format;
```

```
int fscanf (Stream, Format [, Pointer, ... ] )  
FILE * Stream;  
const char *Format;
```

```
int sscanf (String, Format [, Pointer, ... ] )  
const char * String, *Format;
```

```
int wscanf (wcs, Format [, Pointer, ... ] )  
const wchar_t * wcs  
const char *Format;
```

Description

The **scanf**, **fscanf**, **sscanf**, and **wscanf** 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 <i>Stream</i> parameter.
sscanf	Reads from the character string specified by the <i>String</i> parameter.
wscanf	Reads from the wide character string specified by the <i>wcs</i> parameter.

The **scanf**, **fscanf**, **sscanf**, and **wscanf** 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

<i>wcs</i>	Specifies the wide-character string to be read.
<i>Stream</i>	Specifies the input stream.
<i>String</i>	Specifies input to be read.
<i>Pointer</i>	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.
 - ll** 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.
 - A conversion code that specifies the type of conversion to be applied.

The conversion specification takes the form:

```
%[*][width][size]convcode
```

Format (Continued)

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 %% (two 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 **strtod** 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 **wsscanf** 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 sequence is converted to a sequence of wide character codes in the same manner as the **mbstowcs** subroutine. The **wsscanf** 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 *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 **wsscanf** 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 *n*th 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 368) 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 **wscanf** 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 **wscanf** 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** (“sigaction, sigvec, or signal Subroutine” on page 135) 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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **atof**, **atoff**, **strtod**, or **strtof** subroutine, **fread** subroutine, **getc**, **fgetc**, **getchar**, or **getw** subroutine, **gets** or **fgets** subroutine, **getwc**, **fgetwc**, or **getwchar** subroutine, **mbstowcs** subroutine, **mbtowc** subroutine, **printf**, **fprintf**, **sprintf**, **wprintf**, **vprintf**, **vfprintf**, **vsprintf**, or **vwsprintf** subroutine, **setlocale** (“setlocale Subroutine” on page 107) subroutine, **strtol**, **strtoul**, **atol**, or **atoi** (“strtol, strtoul, strtoll, strtoull, atol, or atoi Subroutine” on page 269) subroutine, **ungetc** (“ungetc or ungetwc Subroutine” on page 368) subroutine, **wcstombs** (“wcstombs Subroutine” on page 405) subroutine.

Input and Output Handling Programmer’s Overview, National Language Support Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

sched_yield Subroutine

Purpose

Yield processor.

Library

Standard Library (**libc.a**)

Syntax

```
#include <sched.h>
int sched_yield (void) ;
```

Description

The **sched_yield** function forces the running thread to relinquish the processor until it again becomes the head of its thread list. It takes no arguments.

Return Values

The **sched_yield** function returns 0 if it completes successfully, or it returns a value of -1 and sets `errno` to indicate the error.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) subroutine.

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 sellist * ReadList, *WriteList, *ExceptList;
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 5.1 Communications Programming Concepts*.

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 by Version 3 of the operating system 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 $(1 \ll (n \bmod \text{bits}))$ 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 `msgids` 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 **NMSGs** 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 **NMSGs**(*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.

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.
EINTR	A signal was caught during the select subroutine and the signal handler was installed with an indication that subroutines are not to be restarted.
EINVAL	One of the parameters to the select subroutine contained a value that is not valid.
EFAULT	The <i>ReadList</i> , <i>WriteList</i> , <i>ExceptList</i> , or <i>TimeOut</i> parameter points to a location outside of the address space of the process.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

The **select** subroutine is also supported for compatibility with previous releases of this operating system and with BSD systems.

Related Information

The **poll** subroutine.

The Input and Output Handling Programmer's Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

semctl Subroutine

Purpose

Controls semaphore operations.

Library

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

Syntax

```
#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** (“semop Subroutine” on page 96) 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:

Command	Return Value
GETVAL	Returns the value of the <i>semval</i> field.
GETPID	Returns the value of the <i>sempid</i> field.
GETNCNT	Returns the value of the <i>semcnt</i> field.
GETZCNT	Returns the value of the <i>semzcnt</i> field.
All Others	Return a value of 0.

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 <i>SemaphoreID</i> parameter is not a valid semaphore identifier.
EINVAL	The <i>SemaphoreNumber</i> parameter is less than 0 or greater than or equal to the sem_nsems value.
EINVAL	The <i>Command</i> parameter is not a valid command.
EACCES	The calling process is denied permission for the specified operation.
ERANGE	The <i>Command</i> 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.
EPERM	The <i>Command</i> parameter is equal to the IPC_RMID 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 <code>sem_perm.uid</code> or <code>sem_perm.cuid</code> field in the data structure associated with the <i>SemaphoreID</i> parameter.
EFAULT	The <i>arg.buf</i> or <i>arg.array</i> parameter points outside of the allocated address space of the process.
ENOMEM	The system does not have enough memory to complete the subroutine.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **semget** (“semget Subroutine”) subroutine, **semop** (“semop Subroutine” on page 96) subroutine.

semget Subroutine

Purpose

Gets a set of semaphores.

Library

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

Syntax

```
#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** (“semctl Subroutine” on page 90) 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 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

<i>Key</i>	Specifies either the IPC_PRIVATE value or an IPC key constructed by the ftok subroutine (or a similar algorithm).
<i>NumberOfSemaphores</i>	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_I** 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **ftok** subroutine, **semctl** (“semctl Subroutine” on page 90) subroutine, **semop** (“semop Subroutine” on page 96) subroutine.

The **mode.h** file.

semop Subroutine

Purpose

Performs semaphore operations.

Library

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

Syntax

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

```
int semop (SemaphoreID, SemaphoreOperations, NumberOfSemaphoreOperations)
int SemaphoreID;
struct sembuf * SemaphoreOperations;
size_t 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 **usr/include/sys/sem.h** 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 135) 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

<i>SemaphoreID</i>	Specifies the semaphore identifier.
<i>NumberOfSemaphoreOperations</i>	Specifies the number of structures in the array.
<i>SemaphoreOperations</i>	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 <i>SemaphoreID</i> parameter is not a valid semaphore identifier.
EFBIG	The <i>sem_num</i> value is less than 0 or it is greater than or equal to the number of semaphores in the set associated with the <i>SemaphoreID</i> parameter.
E2BIG	The <i>NumberOfSemaphoreOperations</i> 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 <i>sem_flg</i> 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 <i>SemaphoreOperations</i> parameter points outside of the address space of the process.
EINTR	A signal interrupted the semop subroutine.
EIDRM	The semaphore identifier <i>SemaphoreID</i> parameter has been removed from the system.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **exec** subroutine, **exit** subroutine, **fork** subroutine, **semctl** (“semctl Subroutine” on page 90) subroutine, **semget** (“semget Subroutine” on page 93) subroutine, **sigaction** (“sigaction, sigvec, or signal Subroutine” on page 135) subroutine.

setacldb or endacldb Subroutine

Purpose

Opens and closes the SMIT ACL database.

Library

Security Library (**libc.a**)

Syntax

```
#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 **getgrpacclattr** 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 **getgrpacltr**, **nextgrpacl**, or **putgrpacltr** subroutine, **getusracltr**, **nextusracl**, or **putusracltr** subroutine.

setauditostdb or endauditostdb Subroutine

Purpose

Opens and closes the host identifier auditing file.

Library

Security Library (**libc.a**)

Syntax

```
#include <usersec.h>
int setauditostdb (int Mode);
int endauditostdb (void;
```

Description

These functions can be used to open and close access to the host auditing information database. Programs that call either the **getaudithostattr** or **putaudithostattr** subroutine, call **setaudithostdb** to open the host database and the **endaudithostdb** subroutine to close the host database.

The **setaudithostdb** subroutine opens the host 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 host database when this count goes to 0. Any uncommitted changed data is lost.

Parameter

Mode

Specifies the mode of the open. This parameter may contain one or more of the following values defined in the **usersec.h** files:

S_READ

Specifies read access.

S_WRITE

Specifies write access.

Return Values

On successful completion, the **setaudithostdb** or **endaudithostdb** subroutine returns 0. If unsuccessful, the subroutine returns non-zero.

Error Codes

The **setaudithostdb** or **endaudithostdb** subroutine fails if the following is true:

EINVAL If *Mode* is not one of the valid values.

Related Information

The **auditmerge** command, **auditpr** command, **auditselect** command, **auditstream** command.

The **auditread** subroutine, **getaudithostattr**, **IDtohost**, **hosttoID**, **nexthost** or **putaudithostattr** subroutine.

setbuf, setvbuf, setbuffer, or setlinebuf Subroutine

Purpose

Assigns buffering to a stream.

Library

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

Syntax

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

```
void setbuf ( Stream, Buffer )
```

```
FILE *Stream;
```

```
char *Buffer;
```

```

int setvbuf (Stream, Buffer, Mode, Size)
FILE *Stream;
char *Buffer;
int Mode;
size_t Size;
void setbuffer (Stream, Buffer, Size)
FILE *Stream;
char *Buffer;
size_t Size;
void setlinebuf (Stream)
FILE *Stream;

```

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:

```
char buf[BUFSIZ];
```

For the **setvbuf** subroutine, the *Mode* parameter determines how the *Stream* parameter is buffered:

```

_IIOBF    Causes input/output to be fully buffered.
_IIOBFB   Causes output to be line-buffered. The buffer is flushed when a new line is written, the buffer is full, or
input is requested.
_IIOBNBF  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.

Parameters

<i>Stream</i>	Specifies the input/output stream.
<i>Buffer</i>	Points to a character array.
<i>Mode</i>	Determines how the <i>Stream</i> parameter is buffered.
<i>Size</i>	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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

The **setbuffer** and **setlinebuf** subroutines are included for compatibility with Berkeley System Distribution (BSD).

Related Information

The **fopen**, **freopen**, or **fdopen** subroutine, **fread** subroutine, **getc**, **fgetc**, **getchar**, or **getw** subroutine, **getwc**, **fgetwc**, or **getwchar** 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 5.1 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%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.

Implementation Specifics

The **setcsmap** function is part of Base Operating System (BOS) Runtime.

Related Information

The **setmaps** command.

The **setmaps** file format.

tty Subsystem Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

setgid, setrgid, setegid, or setregid Subroutine

Purpose

Sets the process group IDs.

Library

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

Syntax

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

```
int setgid (GID)
gid_t GID;
```

```
int setrgid (RGID)
gid_t RGID;
```

```
int setegid (EGID)
gid_t EGID;
```

```
int setregid (RGID, EGID)
gid_t RGID;
gid_t EGID;
```

Description

The **setgid**, **setrgid**, **setegid**, and **setregid** 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 <i>GID</i> parameter. Otherwise, the process effective group ID is reset if the <i>GID</i> 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: <ul style="list-style-type: none">• The <i>EGID</i> parameter is equal to either the current real or saved group IDs.• The <i>EGID</i> 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 <i>RGID</i> and <i>EGID</i> parameters can have one of the following relationships: <i>RGID</i> != <i>EGID</i> If the <i>EGID</i> parameter is equal to either the process's real or saved group IDs, the process effective group ID is set to the <i>EGID</i> parameter. Otherwise, the EPERM error code is returned. <i>RGID</i> = <i>EGID</i> If the effective user ID of the process is the root user, the process's real and effective group IDs are set to the <i>EGID</i> parameter. If the <i>EGID</i> parameter is equal to the process's real or saved group IDs, the process effective group ID is set to <i>EGID</i> . Otherwise, the EPERM error code is returned.

The **setegid**, **setrgid**, and **setregid** subroutines are thread-safe.

Parameters

<i>GID</i>	Specifies the value of the group ID to set.
<i>RGID</i>	Specifies the value of the real group ID to set.
<i>EGID</i>	Specifies the value of the effective group ID 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 either the **setgid** or **setegid** subroutine fails, one or more of the following are returned:

EPERM	Indicates the process does not have appropriate privileges and the <i>GID</i> or <i>EGID</i> parameter is not equal to either the real or saved group IDs of the process.
EINVAL	Indicates the value of the <i>GID</i> or <i>EGID</i> parameter is invalid.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

The operating system does not support **setuid** (“setuid, setruuid, seteuid, or setreuid Subroutine” on page 122) or **setgid** shell scripts.

Related Information

The **getgid** subroutine, **getgroups** subroutine, **setgroups** (“setgroups Subroutine”) subroutine, **setuid** (“setuid, setruuid, seteuid, or setreuid Subroutine” on page 122) subroutine.

The **setgroups** command.

List of Security and Auditing Subroutines, Subroutines Overview in *AIX 5L Version 5.1 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

```
#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

<i>GroupIDSet</i>	Pointer to the array of group IDs to be established.
<i>NumberGroups</i>	Indicates the number of entries in the <i>GroupIDSet</i> 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 <i>NumberGroups</i> and <i>GroupIDSet</i> parameters specify an array that is partially or completely outside of the process' allocated address space.
EINVAL	The <i>NumberGroups</i> parameter is greater than the NGROUPS_MAX value.
EPERM	A group ID in the <i>GroupIDSet</i> parameter is not presently in the supplementary group ID, and the invoker does not have root user authority.

Security

Auditing Events:

Event	Information
PROC_SetGroups	NumberGroups, GroupIDSet

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **getgid** subroutine, **getgroups** subroutine, **initgroups** subroutine, **setgid** (“setgid, setrgid, setegid, or setregid Subroutine” on page 103) subroutine.

List of Security and Auditing Subroutines and Subroutines Overview in *AIX 5L Version 5.1 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

```
#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.

Parameters

Context Specifies an address for a **jmp_buf** structure.
Value Indicates any integer value.

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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

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.

Related Information

The **sigsetjmp** or **siglongjmp** (“sigsetjmp or siglongjmp Subroutine” on page 156) subroutine.

Subroutines Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

setlocale Subroutine

Purpose

Changes or queries the program’s entire current locale or portions thereof.

Library

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

Syntax

```
#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 5.1 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:

Language Territory Table			
<i>Locale Value</i>	Language	Territory	Code Set
Ar_AA	Arabic	Arabic Countries	IBM-1046
ar_AA	Arabic	Arabic Countries	ISO8859-6
bg_BG	Bulgarian	Bulgaria	ISO8856-5
cs_CZ	Czech	Czech Republic	ISO8859-2
Da_DK	Danish	Denmark	IBM-850
da_DK	Danish	Denmark	ISO8859-1
De_CH	German	Switzerland	IBM-850
de_CH	German	Switzerland	ISO8859-1
De_DE	German	Germany	IBM-850
de_DE	German	Germany	ISO8859-1
el_GR	Greek	Greece	ISO8859-7
En_GB	English	Great Britain	IBM-850
en_GB	English	Great Britain	ISO8859-1
En_US	English	United States	IBM-850
en_US	English	United States	ISO8859-1
Es_ES	Spanish	Spain	IBM-850
es_ES	Spanish	Spain	ISO8859-1
Fi_FI	Finnish	Finland	IBM-850
fi_FI	Finnish	Finland	ISO8859-1
Fr_BE	French	Belgium	IBM-850
fr_BE	French	Belgium	ISO8859-1
Fr_CA	French	Canada	IBM-850
fr_CA	French	Canada	ISO8859-1
Fr_FR	French	France	IBM-850
fr_FR	French	France	ISO8859-1
Fr_CH	French	Switzerland	IBM-850
fr_CH	French	Switzerland	ISO8859-1
hr_HR	Croatian	Croatia	ISO8859-2
hu_HU	Hungarian	Hungary	ISO8859-2
Is_IS	Icelandic	Iceland	IBM-850

is_IS	Icelandic	Iceland	ISO8859-1
It_IT	Italian	Italy	IBM-850
it_IT	Italian	Italy	ISO8859-1
Iw_IL	Hebrew	Israel	IBM-856
iw_IL	Hebrew	Israel	ISO8859-8
Ja_JP	Japanese	Japan	IBM-943
ja_JP	Japanese	Japan	IBM-eucJP
ko_KR	Korean	Korea	IBM-eucKR
mk_MK	Macedonian	Former Yugoslav Republic of Macedonia	ISO8859-5
NI_BE	Dutch	Belgium	IBM-850
nl_BE	Dutch	Belgium	ISO8859-1
NI_NL	Dutch	Netherlands	IBM-850
nl_NL	Dutch	Netherlands	ISO8859-1
No_NO	Norwegian	Norway	IBM-850
no_NO	Norwegian	Norway	ISO8859-1
pl_PL	Polish	Poland	ISO8859-2
Pt_PT	Portuguese	Portugal	IBM-850
pt_PT	Portuguese	Portugal	ISO8859-1
ro_RO	Romanian	Romania	ISO8859-2
ru_RU	Russian	Russia	ISO8859-5
sh_SP	Serbian Latin	Yugoslavia	ISO8859-2
sl_SI	Slovene	Slovenia	ISO8859-2
sk_SK	Slovak	Slovakia	ISO8859-2
sr_SP	Serbian Cyrillic	Yugoslavia	ISO8859-5
Zh_CN	Simplified Chinese	PRC	GBK
Sv_SE	Swedish	Sweden	IBM-850
sv_SE	Swedish	Sweden	ISO8859-1
tr_TR	Turkish	Turkey	ISO8859-9
zh_TW	Chinese (trad)	Republic of China	IBM-eucTW

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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **localeconv** subroutine, **nl_langinfo** subroutine, **rpmatch** (“rpmatch Subroutine” on page 40) subroutine.

The **localedef** command.

National Language Support Overview for Programming and Understanding Locale Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Subroutines Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

setpcred Subroutine

Purpose

Sets the current process credentials.

Library

Security Library (**libc.a**)

Syntax

```
#include <usersec.h>
```

```
int setpcred ( User, Credentials)
char **Credentials;
char *User;
```

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

RLIMIT_FSIZE=%d
Process soft file size

RLIMIT_DATA=%d
Process soft data segment size

RLIMIT_STACK=%d
Process soft stack segment size

RLIMIT_CORE=%d
Process soft core file size

RLIMIT_RSS=%d
Process soft resident set size

RLIMIT_CORE_HARD=%d
Process hard core file size

RLIMIT_CPU_HARD=%d
Process hard CPU limit

RLIMIT_DATA_HARD=%d
Process hard data segment size

RLIMIT_FSIZE_HARD=%d
Process hard file size

RLIMIT_RSS_HARD=%d
Process hard resident set size

RLIMIT_STACK_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.

Resource	Hard	Soft
RLIMIT_CORE	unlimited	%d
RLIMIT_CPU	%d	%d
RLIMIT_DATA	unlimited	%d
RLIMIT_FSIZE	%d	%d
RLIMIT_RSS	unlimited	%d
RLIMIT_STACK	unlimited	%d

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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **auditwrite** subroutine, **ckuseracct** subroutine, **ckuserID** subroutine, **getpcred** subroutine, **getpenv** subroutine, **setpenv** (“setpenv Subroutine”) subroutine.

List of Security and Auditing Subroutines, Subroutines Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

setpenv Subroutine

Purpose

Sets the current process environment.

Library

Security Library (**libc.a**)

Syntax

```
#include <usersec.h>
```

```
int setpenv ( User, Mode, Environment, Command)
char *User;
int Mode;
char **Environment;
char *Command;
```

Description

The **setpenv** 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 **setpenv** 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 **execv** 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 8 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/enviro**n 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.

PENV_DELTA

The existing user-state and system-state environment variables are preserved and the variables defined in the *Environment* parameter are added.

PENV_RESET

The existing environment is cleared and totally replaced by the content of the *Environment* parameter.

PENV_KLEEN

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.

PENV_NOPROF

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:

PENV_ARGV 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.

User 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.

Return Values

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.

Error Codes

The **setpenv** subroutine fails if one or more of the following are true:

- EINVAL** The *Mode* parameter contains values other than **PENV_INIT**, **PENV_DELTA**, **PENV_RESET**, or **PENV_ARGV**.
- EINVAL** The *Mode* parameter contains more than one of **PENV_INIT**, **PENV_DELTA**, or **PENV_RESET** values.
- EINVAL** The *Environment* parameter is neither null nor empty, and does not contain a valid environment string.
- EPERM** 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **execl**, **execv**, **execle**, **execve**, **execlp**, **execvp**, or **exec** subroutine, **getpenv** subroutine, **setpcrd** (“setpcrd Subroutine” on page 111) subroutine.

The **login** command, **su** command.

List of Security and Auditing Subroutines, Subroutines Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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>
```

```
int setpgid ( ProcessID, ProcessGroupID)
pid_t ProcessID, ProcessGroupID;
int 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.

Parameters

<i>ProcessID</i>	Specifies the process whose process group ID is to be changed.
<i>ProcessGroupID</i>	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 <i>ProcessGroupID</i> 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 <i>ProcessID</i> parameter is a session leader.
EPERM	The value of the <i>ProcessID</i> 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 <i>ProcessGroupID</i> parameter is valid, but does not match the process ID of the process indicated by the <i>ProcessID</i> parameter. There is no process with a process group ID that matches the value of the <i>ProcessGroupID</i> parameter in the same session as the calling process.
ESRCH	The value of the <i>ProcessID</i> parameter does not match the process ID of the calling process of a child process of the calling process.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

In BSD systems, the **setpgrp** subroutine is defined with two parameters, as follows:

```
int setpgrp (ProcessID, ProcessGroup)
int ProcessID, ProcessGroup;
```

BSD systems set the process group to the value specified by the *ProcessGroup* parameter. If the *ProcessID* value is 0, the call applies to the current process. In AIX 3.1, this version of the **setpgrp** subroutine must be compiled with the Berkeley Compatibility Library (**libbsd.a**) and is implemented as a call to the **setpgid** subroutine. The restrictions that apply to the **setpgid** subroutine also apply to the **setpgrp** subroutine.

BSD systems set the process group to the value specified by the *ProcessGroup* parameter. If the *ProcessID* value is 0, the call applies to the current process. In the Version 3 Operating System, this version of the **setpgrp** subroutine must be compiled with the Berkeley Compatibility Library (**libbsd.a**) and is implemented as a call to the **setpgid** subroutine. The restrictions that apply to the **setpgid** subroutine also apply to the **setpgrp** subroutine.

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 ( ProcessID, Priority)
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

<i>ProcessID</i>	Specifies the process ID. If this value is 0 then the current process scheduling priority is set to a constant.
<i>Priority</i>	Specifies the scheduling priority for the process. A lower number value designates a higher scheduling priority. The <i>Priority</i> parameter must be in the range PRIORITY_MIN < <i>Priority</i> < 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 <i>Priority</i> 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 <i>ProcessID</i> parameter.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **getpri** subroutine.

Performance-Related Subroutines in *AIX 5L Version 5.1 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:

Modes	File
rw	/etc/security/passwd
rw	/etc/passwd

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

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

```
#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

```
#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, `(pid_t) -1` is returned and the `errno` global variable is set to indicate the error.

Error Codes

The `setsid` subroutine is unsuccessful if the following is true:

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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The `fork` subroutine, `getpid`, `getpgrp`, or `getppid` subroutine, `setpgid` (“`setpgid` or `setpgrp` Subroutine” on page 117) subroutine, `setpgrp` (“`setpgid` or `setpgrp` Subroutine” on page 117) subroutine.

setuid, setruid, seteuid, or setreuid Subroutine

Purpose

Sets the process user IDs.

Library

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

Syntax

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

```
int setuid (UID)
uid_t UID;
```

```
int setruid (RUID)
uid_t RUID;
```

```
int seteuid (EUID)
uid_t EUID;
```

```
int setreuid (RUID, EUID)
uid_t RUID;
uid_t EUID;
```

Description

The **setuid**, **setruid**, **seteuid**, and **setreuid** subroutines reset the process user IDs. The following semantics are supported:

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 <i>UID</i> parameter. Otherwise, the process effective user ID is reset if the <i>UID</i> parameter specifies either the current real or saved user IDs.
seteuid	The process effective user ID is reset if the <i>UID</i> 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.
setruid	The EPERM error code is always returned. Processes cannot reset only their real user IDs.
setreuid	The <i>RUID</i> and <i>EUID</i> parameters can have the following two possibilities: <i>RUID</i> != <i>EUID</i> If the <i>EUID</i> parameter specifies either the process's real or saved user IDs, the process effective user ID is set to the <i>EUID</i> parameter. Otherwise, the EPERM error code is returned. <i>RUID</i> = <i>EUID</i> If the process effective user ID is the root user, the process's real and effective user IDs are set to the <i>EUID</i> parameter. Otherwise, the EPERM error code is returned.

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.

Parameters

<i>UID</i>	Specifies the user ID to set.
<i>EUID</i>	Specifies the effective user ID to set.
<i>RUID</i>	Specifies the real user ID to set.

Return Values

Upon successful completion, the **setuid**, **seteuid**, and **setreuid** 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**, and **setreuid** subroutines are unsuccessful if either of the following is true:

- EINVAL** The value of the *UID* or *EUID* parameter is not valid.
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.

Implementation Specifics

The operating system does not support **setuid** or **setgid** (“setgid, setrgid, setegid, or setregid Subroutine” on page 103) shell scripts.

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **getuid** or **geteuid** subroutine, **setgid** (“setgid, setrgid, setegid, or setregid Subroutine” on page 103) subroutine.

List of Security and Auditing Subroutines and Subroutines Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

setuserdb or enduserdb Subroutine

Purpose

Opens and closes the user database.

Library

Security Library (**libc.a**)

Syntax

```
#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.

Note: These subroutines are not safe for use with multiple threads. To call one of these subroutines from a threaded application, enclose the call with the **_libs_rmutex** lock. See “Making a Subroutine Safe for Multiple Threads” in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs* for more information about this lock.

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:

Modes	File
rw	/etc/passwd
rw	/etc/group
rw	/etc/security/user
rw	/etc/security/limits
rw	/etc/security/group
rw	/etc/security/environ

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **getgroupattr** subroutine, **getuserattr** subroutine, **getuserpw** subroutine, **setpwnb** (“setpwnb or endpwnb Subroutine” on page 119) subroutine.

List of Security and Auditing Subroutines and Subroutines Overview in *AIX 5L Version 5.1 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

Value Specifies a 4-byte value to store into memory.
Buffer Points to a location in memory.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **ar** command, **dump** command.

The **ar** file format, **a.out** file format.

Subroutines Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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 **openx** subroutine) to the address space of the calling process.

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.

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, although for segments larger than 256MB in size the environment variable **EXTSHM=ON** is ignored. 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 is at page boundaries - a multiple of **SHMLBA_EXTSHM** bytes. For segments larger than 256MB in size, 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 *mapped* regions.

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 a 256MB area of the address space irrespective 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 a 256MB area of the address space irrespective 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.

Parameters

<i>SharedMemoryID</i>	Specifies an identifier for the shared memory segment.
<i>SharedMemoryAddress</i>	Identifies the segment or file attached at the address specified by the <i>SharedMemoryAddress</i> parameter, as follows: <ul style="list-style-type: none">• If the <i>SharedMemoryAddress</i> parameter is not equal to 0, and the SHM_RND flag is set in the <i>SharedMemoryFlag</i> parameter, the segment or file is attached at the next lower segment boundary. This address is given by $(\text{SharedMemoryAddress} - (\text{SharedMemoryAddress} \bmod \text{SHMLBA_EXTSHM}))$ if environment variable EXTSHM=ON or SHMLBA if not). SHMLBA specifies the low boundary address multiple of a segment.• If the <i>SharedMemoryAddress</i> parameter is not equal to 0 and the SHM_RND flag is not set in the <i>SharedMemoryFlag</i> parameter, the segment or file is attached at the address given by the <i>SharedMemoryAddress</i> 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.
<i>SharedMemoryFlag</i>	Specifies 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 <i>SharedMemoryID</i> 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 <i>SharedMemoryAddress</i> 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.

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 <i>SharedMemoryID</i> parameter is not a valid shared memory identifier.
EINVAL	The <i>SharedMemoryAddress</i> parameter is not equal to 0, and the value of (<i>SharedMemoryAddress</i> - (<i>SharedMemoryAddress</i> modulo SHMLBA_EXTSHM if the environment variable EXTSHM=ON or SHMLBA if not) points outside the address space of the process.
EINVAL	The <i>SharedMemoryAddress</i> parameter is not equal to 0, the SHM_RND flag is not set in the <i>SharedMemoryFlag</i> parameter, and the <i>SharedMemoryAddress</i> 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.
ENOMEM	The requested address and length crosses a segment boundary. This is not supported when the environment variable EXTSHM=ON .

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **exec** subroutine, **exit** subroutine, **fclean** subroutine, **fork** subroutine, **fsync** subroutine, **mmap** subroutine, **munmap** subroutine, **openx** subroutine, **truncate** subroutine, **readvx** subroutine, **shmctl** subroutine, **shmdt** subroutine, **shmget** subroutine, **writevx** subroutine.

The **ipcs** command and **ipcrm** command.

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 shmids * 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 **shmids** structure. The **shmids** structure is defined in the **sys/shm.h** file.

Command

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.cuid` or `shm_perm.uid` field in the **shmid_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 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.

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**.
- 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

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 5.1 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **exec** subroutine, **exit** subroutine, **fork** subroutine, **fsync** subroutine, **mmap** subroutine, **munmap** subroutine, **shmat** subroutine, **shmctl** subroutine, **shmget** subroutine.

The **ipcs** command and **ipcrm** command.

List of Memory Manipulation Services, Subroutines Overview, Understanding Memory Mapping in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

shmget Subroutine

Purpose

Gets shared memory segments.

Library

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

Syntax

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

```
int shmget (Key, Size, SharedMemoryFlag)
```

```
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 and 131072 for AIX 4.3.2 and following.

Parameters

<i>Key</i>	Specifies either the IPC_PRIVATE value or an IPC key constructed by the ftok subroutine (or by a similar algorithm).
<i>Size</i>	Specifies the number of bytes of shared memory required.
<i>SharedMemoryFlag</i>	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 via 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**). 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 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 <i>Key</i> parameter, but operation permission as specified by the low-order 9 bits of the <i>SharedMemoryFlag</i> parameter is not granted. |
| EEXIST | A shared memory identifier exists for the <i>Key</i> parameter, and both the IPC_CREAT and IPC_EXCL flags are set in the <i>SharedMemoryFlag</i> parameter. |
| EINVAL | A shared memory identifier does not exist and the <i>Size</i> parameter is less than the system-imposed minimum or greater than the system-imposed maximum. |
| EINVAL | A shared memory identifier exists for the <i>Key</i> parameter, but the size of the segment associated with it is less than the <i>Size</i> parameter, and the <i>Size</i> parameter is not equal to 0. |
| ENOENT | A shared memory identifier does not exist for the <i>Key</i> parameter, and the IPC_CREAT flag is not set in the <i>SharedMemoryFlag</i> 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. |

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

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 5.1 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:

Member Type	Member Name	Description
void(*) (int)	sa_handler	SIG_DFL, SIG_IGN or pointer to a function.
sigset_t	sa_mask	Additional set of signals to be blocked during execution of signal-catching function.
int	sa_flags	Special flags to affect behaviour of signal.
void(*) (int, siginfo_t *, void *)	sa_sigaction	Signal-catching function.

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** ("read, readx, readv, readvx, or pread Subroutine" on page 9)
- **write, writex, writev, or writevx** ("write, writex, writev, writevx or pwrite Subroutines" on page 449)
- **ioctl orioctlx**
- **fcntl, lockf, or flock**
- **wait, wait3, orwaitpid** ("wait, waitpid, wait3, or wait364 Subroutine" on page 385)

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: `void func(int signo);`

where `signo` is the only argument to the signal catching function. In this case the `sa_handler` member must be used to describe the signal catching function and the application must not modify the `sa_sigaction` member. If **SA_SIGINFO** is set and the signal is caught, the signal-catching function will be entered as: `void func(int signo, siginfo_t * info, void * context);` where two additional arguments are passed to the signal catching function. 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 non-zero, it contains an error number identifying the condition that caused the signal to be generated. The **si_code** member contains a code identifying the cause of the signal. If the value of **si_code** is less than or equal to 0, then the signal was generated by a process and **si_pid** and **si_uid** respectively indicate the process ID and the real user ID of the sender. The **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**. 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 and the **SA_NOCLDSTOP** flag is not set in **sa_flags**, and the implementation supports the SIGCHLD signal, then a SIGCHLD signal will be generated for the calling process whenever any of its child processes stop. If **sig** is SIGCHLD and the **SA_NOCLDSTOP** flag is set in **sa_flags**, then the implementation will not generate a SIGCHLD signal in this way. When a signal is caught by a signal-catching function installed by **sigaction**, a new signal mask is calculated and installed for the duration of the signal-catching function (or until a call to either **sigprocmask** or **sigsuspend** is made). This mask is formed by taking the union of the current signal mask and the value of the **sa_mask** for the signal being delivered unless **SA_NODEFER** or **SA_RESETHAND** is set, and then including the signal being delivered. If and when the user's signal handler returns normally, the original signal mask is restored. Once an action is installed for a specific signal, it remains installed until another action is explicitly requested (by another call to **sigaction**()), until the **SA_RESETHAND** flag causes resetting of the handler, or until one of the exec functions is called. If the previous action for **sig** had been established by **signal**, the values of the fields returned in the structure pointed to by **oact** are unspecified, and in particular **oact->sa_handler** is not necessarily the same value passed to **signal**. However, if a pointer to the same structure or a copy thereof is passed to a subsequent call to **sigaction** via the **act** argument, handling of the signal will be as if the original call to **signal** were repeated. If **sigaction** fails, no new signal handler is installed. It is unspecified whether an attempt to set the action for a signal that cannot be caught or ignored to **SIG_DFL** is ignored or causes an error to be returned with **errno** set to **EINVAL**.

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**.

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+)

SIGTTIN
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 a ! (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 a @ (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 +, 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-catching function specified by the pointer to function. The signal-handler subroutine can be declared as follows:

```
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-catching 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-catching 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 **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 **longjmp** subroutine. When the **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 **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 **sigaction** subroutine), or until one of the **exec**

subroutines is called. An exception to this is when the **SA_OLDSTYLE** bit is set. In this case the action of a caught signal gets set to the **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:

_exit

access

alarm

cfgetispeed

cfgetospeed

cfsetispeed

cfsetospeed

chdir

chmod

chown

close

creat

dup

dup2

exec

execle

execve

fcntl

fork

fpathconf

fstat

getegid

geteuid

getgid

getgroups

getpgrp

getpid

getppid

getuid

kill

link
lseek
mkdir
mkfifo
open
pathconf
pause
pipe
raise
read
readx
rename
rmdir
setgid
setpgid
setpgrp
setsid
setuid
sigaction
sigaddset
sigdelset
sigemptyset
sigismember
signal
sigpending
sigprocmask
sigsuspend
sleep
stat
statx
sysconf
tcdrain
tcflow
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.

OAction

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 *OAction* 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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **acct** subroutine, **_exit**, **exit**, or **atexit** subroutine, **getinterval**, **incinterval**, **absinterval**, **resinc**, **resabs**, **alarm**, **ualarm**, **getitimer**, or **setitimer** subroutine, **getrlimit**, **setrlimit**, or **vlimit** subroutine, **kill**

subroutine, **longjmp** or **setjmp** (“setjmp or longjmp Subroutine” on page 106) subroutine, **pause** subroutine, **ptrace** subroutine, **sigpause** or **sigsuspend** (“sigsuspend or sigpause Subroutine” on page 158) subroutine, **sigprocmask**, **sigsetmask**, or **sigblock** (“sigprocmask, sigsetmask, or sigblock Subroutine” on page 151) subroutine, **sigstack** (“sigstack Subroutine” on page 157) subroutine, **sigwait** (“sigwait Subroutine” on page 161) subroutine, **umask** (“umask Subroutine” on page 364) subroutine, **wait**, **waitpid**, or **wait3** (“wait, waitpid, wait3, or wait364 Subroutine” on page 385) subroutine.

The **kill** command.

The **core** file.

Signal Management in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs* provides more information about signal management in multi-threaded processes.

sigaltstack Subroutine

Purpose

Allows a process 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 process to define and examine the state of an alternate stack for signal handlers. Signals that have been explicitly declared to execute on the alternate stack will be delivered on the alternate stack.

If *ss* is not 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 than **SS_DISABLE**.
ENOMEM The size of the alternate stack area is less than **MINSIGSTKSZ**.
EPERM An attempt was made to modify an active stack.

Related Information

The **sigaction** (“sigaction, sigvec, or signal Subroutine” on page 135) subroutine, **sigsetjmp** (“sigsetjmp or siglongjmp Subroutine” on page 156) 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;
```

```
int sigfillset (Set)  
sigset_t *Set;
```

```
int sigaddset (Set, SignalNumber)  
sigset_t *Set;  
int SignalNumber;
```

```

int sigdelset (Set, SignalNumber)
sigset_t *Set;
int SignalNumber;

int sigismember (Set, SignalNumber)
sigset_t *Set;
int SignalNumber;

```

Description

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

<i>Set</i>	Specifies the signal set.
<i>SignalNumber</i>	Specifies the individual signal.

Examples

To generate and use a signal mask that blocks 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);
sigaddset(newset, SIGINT);
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:

EINVAL The value of the *SignalNumber* parameter is not a valid signal number.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **sigaction**, **sigvec**, or **signal** (“sigaction, sigvec, or signal Subroutine” on page 135) subroutine, **sigprocmask** (“sigprocmask, sigsetmask, or sigblock Subroutine” on page 151) subroutine, **sigsuspend** (“sigsuspend or sigpause Subroutine” on page 158) subroutine.

siginterrupt Subroutine

Purpose

Sets restart behavior with respect to signals and subroutines.

Library

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

Syntax

```
int siginterrupt ( Signal, Flag )
    int 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.

Parameters

<i>Signal</i>	Indicates the signal.
<i>Flag</i>	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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

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.

Related Information

The **sigaction** or **sigvec** (“sigaction, sigvec, or signal Subroutine” on page 135) subroutine, **sigpause** (“sigsuspend or sigpause Subroutine” on page 158) subroutine, **sigsetmask** or **sigblock** (“sigprocmask, sigsetmask, or sigblock Subroutine” on page 151) subroutine.

sigpending Subroutine

Purpose

Returns a set of signals that are blocked from delivery.

Library

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

Syntax

```
#include <signal.h>
```

```
int sigpending ( Set )  
sigset_t *Set;
```

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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **sigprocmask** (“sigprocmask, sigsetmask, or sigblock Subroutine” on page 151) subroutine.

sigprocmask, sigsetmask, or sigblock Subroutine

Purpose

Sets the current signal mask.

Library

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

Syntax

```
#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 159) 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.

<i>Set</i>	Specifies the signal set. If the value of the <i>Set</i> 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 <i>Set</i> parameter is null, the value of the <i>How</i> parameter is not significant and the process signal mask is unchanged. Thus, the call can be used to inquire about currently blocked signals.
<i>OSet</i>	If the <i>OSet</i> 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 <i>OSet</i> parameter.
<i>SignalMask</i>	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 <i>How</i> parameter is not equal to one of the defined values.
EFAULT	The user's mask is not in the process address space.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

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);
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **kill** or **killpg** subroutine, **sigaction**, **sigvec**, or **signal** (“sigaction, sigvec, or signal Subroutine” on page 135) subroutine, **sigaddset**, **sigdelset**, **sigemptyset**, **sigfillset**, **sigismember** (“sigemptyset, sigfillset, sigaddset, sigdelset, or sigismember Subroutine” on page 147) subroutine, **sigpause** (“sigsuspend or sigpause Subroutine” on page 158) subroutine, **sigpending** (“sigpending Subroutine” on page 150) subroutine, **sigsuspend** (“sigsuspend or sigpause Subroutine” on page 158) subroutine.

sigset, sighold, sigrelse, or sigignore Subroutine

Purpose

Enhance the signal facility and provide signal management.

Library

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

Syntax

```
#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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **exit** subroutine, **kill** subroutine, **setjmp** (“setjmp or longjmp Subroutine” on page 106) subroutine, **signal** (“sigaction, sigvec, or signal Subroutine” on page 135) subroutine, **wait** (“wait, waitpid, wait3, or wait364 Subroutine” on page 385) subroutine.

sigsetjmp or siglongjmp Subroutine

Purpose

Saves or restores stack context and signal mask.

Library

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

Syntax

```
#include <setjmp.h>
```

```
int sigsetjmp ( Environment, SaveMask)
sigjmp_buf Environment;
int SaveMask;
```

```
void siglongjmp (Environment, Value)
sigjmp_buf Environment;
int Value;
```

Description

The **sigsetjmp** subroutine saves the current stack context, and if the value of the *SaveMask* parameter is not 0, the **sigsetjmp** subroutine also saves the current signal mask of the process as part of the calling environment.

The **siglongjmp** subroutine restores the saved signal mask only if the *Environment* parameter was initialized by a call to the **sigsetjmp** subroutine with a nonzero *SaveMask* parameter argument.

Parameters

<i>Environment</i>	Specifies an address for a sigjmp_buf structure.
<i>SaveMask</i>	Specifies the flag used to determine if the signal mask is to be saved.
<i>Value</i>	Specifies the return value from the siglongjmp subroutine.

Return Values

The **sigsetjmp** subroutine returns a value of 0. The **siglongjmp** subroutine returns a nonzero value.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **setjmp** or **longjmp** (“setjmp or longjmp Subroutine” on page 106) subroutine, **sigaction** (“sigaction, sigvec, or signal Subroutine” on page 135) subroutine, **sigprocmask** (“sigprocmask, sigsetmask, or sigblock Subroutine” on page 151) subroutine, **sigsuspend** (“sigsuspend or sigpause Subroutine” on page 158) 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 If the value of the *InStack* parameter is 0 (that is, a null pointer), the signal stack state is not set. 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **longjmp** (“setjmp or longjmp Subroutine” on page 106) subroutine, **setjmp** (“setjmp or longjmp Subroutine” on page 106) subroutine, **sigaction**, **signal**, or **sigvec** (“sigaction, sigvec, or signal Subroutine” on page 135) 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

```
#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 **sigthreadmask(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 **sigthreadmask** 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**.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

The **sigpause** subroutine is provided for compatibility with older UNIX systems; its function is a subset of the **sigsuspend** subroutine.

Related Information

The **pause** subroutine, **sigprocmask** (“sigprocmask, sigsetmask, or sigblock Subroutine” on page 151) subroutine, **sigaction** or **signal** (“sigaction, sigvec, or signal Subroutine” on page 135) subroutine, **sigthreadmask** (“sigthreadmask Subroutine”) subroutine.

Signal Management in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs* provides more information about signal management in multi-threaded processes.

sigthreadmask Subroutine

Purpose

Sets the signal mask of a thread.

Library

Threads Library (**libpthreads.a**)

Syntax

```
#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

<i>how</i>	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 <i>set</i> parameter. SIG_UNBLOCK The resulting set is the intersection of the current set and the complement of the signal set pointed to by the <i>set</i> parameter. SIG_SETMASK The resulting set is the signal set pointed to by the <i>set</i> parameter.
<i>set</i>	Specifies the signal set. If the value of the <i>Set</i> 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 <i>Set</i> parameter is null, the value of the <i>How</i> parameter is not significant and the process signal mask is unchanged. Thus, the call can be used to inquire about currently blocked signals.
<i>old_set</i>	If the <i>old_set</i> 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 <i>old_set</i> 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 <i>set</i> or <i>old_set</i> pointers are not in the process address space.
EINVAL	The value of the <i>how</i> 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:

```
#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);

```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **kill** or **killpg** subroutine, **pthread_kill** subroutine, **sigaction**, **sigvec**, or **signal** (“sigaction, sigvec, or signal Subroutine” on page 135) subroutine, **sigpause** (“sigsuspend or sigpause Subroutine” on page 158) subroutine, **sigpending** (“sigpending Subroutine” on page 150) subroutine, **sigwait** (“sigwait Subroutine”) subroutine, **sigsuspend** (“sigsuspend or sigpause Subroutine” on page 158) subroutine.

Signal Management in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

sigwait Subroutine

Purpose

Blocks the calling thread until a specified signal is received.

Library

Threads Library (**libpthreads.a**)

Syntax

```
#include </usr/include/dce/cma_sigwait.h>
```

```

int sigwait ( set, sig)
const sigset_t *set;
int *sig;

```

Description

The **sigwait** subroutine blocks the calling thread until one of the signal in the signal set *set* is received by the thread. Only asynchronous signals can be waited for.

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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **kill** subroutine, **pthread_kill** subroutine, **sigaction** (“sigaction, sigvec, or signal Subroutine” on page 135) subroutine, **sigthreadmask** (“sigthreadmask Subroutine” on page 159) subroutine.

Signal Management in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs* .

sin, sinl, cos, cosl, tan, or tanl Subroutine

Purpose

Computes the trigonometric functions.

Libraries

IEEE Math Library (**libm.a**)

or System V Math Library (**libmsaa.a**)

Syntax

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

```
double sin ( x )
```

```
double x;
```

```
long double sinl ( x )
```

```
long double x;
```

```
double cos ( x )
```

```
double x;
```

```
long double cosl ( x )
```

```
long double x;
```

```
double tan ( x )
```

```
double x;
```

```
long double tanl ( x )
```

```
long double x;
```

Description

The **sin**, **cos**, and **tan** subroutines return the sine, cosine, and tangent, respectively, of their parameters, which are in radians. The **sinl** subroutine, **cosl** subroutine, and **tanl** subroutine return the same values, but these subroutines take and return numbers of the long double data type.

Parameters

- x* Specifies some double-precision floating-point value. For the **sinl**, **cosl**, and **tanl** subroutines, specifies a long double-precision floating-point value.
- y* Specifies some double-precision floating-point value. For the **sinl** subroutine, **cosl** subroutine, and **tanl** subroutine, specifies a long double-precision floating-point value.

Error Codes

The **sin**, **sinl**, **cos**, **cosl**, **tan**, and **tanl** 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 π 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 π 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**, **cos**, **cosl**, **tan**, and **tanl** subroutines during argument reduction of large arguments.

sin, **cos**, 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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **matherr** subroutine, **sinh**, **sinhl**, **cosh**, **coshl**, **tanh**, or **tanh1** (“sinh, sinhl, cosh, coshl, tanh, or tanhl Subroutine”) subroutines.

Subroutines Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

128-Bit long double Floating-Point Format in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

sinh, sinhl, cosh, coshl, tanh, or tanhl Subroutine

Purpose

Computes hyperbolic functions.

Libraries

IEEE Math Library (**libm.a**)

or System V Math Library (**libmsaa.a**)

Syntax

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

```
double sinh ( x)
double x;
double cosh (x)
double x;
double tanh (x)
double x;
long double sinhl (x)
double x;
long double coshl (x)
double x;
long double tanhl (x)
double x;
```

Description

The **sinh**, **cosh**, and **tanh** subroutines compute the hyperbolic trigonometric functions of their parameters. The **sinhl**, **coshl**, and **tanhl** subroutines compute these functions for parameters expressed in the long double data type.

Note: Compile any routine that uses subroutines from the **libm.a** library with the **-lm** flag. To compile the **tanh.c** file, for example, enter:

```
cc tanh.c -lm
```

Parameters

x Specifies a double-precision floating-point value.

Error Codes

If the correct value overflows, the **sinh**, **sinhl**, **cosh**, and **coshl** 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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **matherr** subroutine, **sin**, **cos**, **tan**, **asin**, **acos**, **atan**, or **atan2** (“sin, sinl, cos, cosl, tan, or tanl Subroutine” on page 162) subroutine.

Subroutines Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

128-Bit long double Floating-Point Format in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

sleep, nsleep or usleep Subroutine

Purpose

Suspends 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

<i>Rqtp</i>	Time interval specified for suspension of execution.
<i>Rmtp</i>	Specifies the time remaining on the interval timer or 0.
<i>Seconds</i>	Specifies time interval in seconds.
<i>Useconds</i>	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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **alarm** subroutine, **pause** subroutine, **sigaction** (“sigaction, sigvec, or signal Subroutine” on page 135) subroutine.

List of Time Data Manipulation Services in *AIX 5L Version 5.1 System Management Concepts: Operating System and Devices*.

Subroutines Overview in *AIX 5L Version 5.1 System Management Guide: Operating System and Devices*.

snprintf Subroutine

Purpose

Print formatted output. `int snprintf(char *s, size_t n, const char *format, . . .);`

Library

Standard library (**libc.a**)

Syntax

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

```
int snprintf (char *s, size_t n, const char *format, . . .) ;
```

Description

The **fprintf** function places output on the named output **stream**. The **printf** function places output on the standard output stream **stdout**. The **sprintf** function places output followed by the null byte, '\0', in consecutive bytes starting at ***s**; it is the user's responsibility to ensure that enough space is available.

snprintf is identical to **sprintf** with the addition of the **n** argument, which states the size of the buffer referred to by **s**.

Each of these functions converts, formats and prints its arguments under control of the **format**. The **format** is a character string, beginning and ending in its initial shift state, if any. The **format** is composed of zero or more directives: **ordinary characters**, which are simply copied to the output stream and conversion specifications, each of which results in the fetching of zero or more arguments. The results are undefined if there are insufficient arguments for the **format**. If the **format** is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

Conversions can be applied to the *n*th argument after the **format** in the argument list, rather than to the next unused argument. In this case, the conversion character % (see below) is replaced by the sequence %*n*\$, where *n* is a decimal integer in the range [1, {NL_ARGMAX}], giving the position of the argument in the argument list. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages (see the EXAMPLES section).

In format strings containing the %*n*\$ form of conversion specifications, numbered arguments in the argument list can be referenced from the format string as many times as required.

In format strings containing the % form of conversion specifications, each argument in the argument list is used exactly once.

All forms of the **fprintf** functions allow for the insertion of a language-dependent radix character in the output string. The radix character is defined in the program's locale (category LC_NUMERIC). In the POSIX locale, or in a locale where the radix character is not defined, the radix character defaults to a period (.).

Each conversion specification is introduced by the % character or by the character sequence %*n*\$, after which the following appear in sequence:

- Zero or more **flags** (in any order), which modify the meaning of the conversion specification.
- An optional minimum field width. If the converted value has fewer bytes than the field width, it will be padded with spaces by default on the left; it will be padded on the right, if the left-adjustment flag (-), described below, is given to the field width. The field width takes the form of an asterisk (*), described below, or a decimal integer.
- An optional precision that gives the minimum number of digits to appear for the d, i, o, u, x and X conversions; the number of digits to appear after the radix character for the e, E and f conversions; the maximum number of significant digits for the g and G conversions; or the maximum number of bytes to be printed from a string in s and S conversions. The precision takes the form of a period (.) followed either by an asterisk (*), described below, or an optional decimal digit string, where a null digit string is treated as 0. If a precision appears with any other conversion character, the behavior is undefined.
- An optional h specifying that a following d, i, o, u, x or X conversion character applies to a type short int or type unsigned short int argument (the argument will have been promoted according to the integral promotions, and its value will be converted to type short int or unsigned short int before printing); an optional h specifying that a following n conversion character applies to a pointer to a type short int argument; an optional l (ell) specifying that a following d, i, o, u, x or X conversion character applies to a type long int or unsigned long int argument; an optional l (ell) specifying that a following n conversion character applies to a pointer to a type long int argument; or an optional L specifying that a following e, E, f, g or G conversion character applies to a type long double argument. If an h, l or L appears with any other conversion character, the behavior is undefined.
- An optional l specifying that a following c conversion character applies to a **wint_t** argument; an optional l specifying that a following s conversion character applies to a pointer to a **wchar_t** argument.
- A conversion character that indicates the type of conversion to be applied.

A field width, or precision, or both, may be indicated by an asterisk (*). In this case an argument of type int supplies the field width or precision. Arguments specifying field width, or precision, or both must appear in that order before the argument, if any, to be converted. A negative field width is taken as a - flag followed by a positive field width. A negative precision is taken as if the precision were omitted. In format strings containing the %*n*\$ form of a conversion specification, a field width or precision may be indicated by the sequence **m*\$, where *m* is a decimal integer in the range [1, {NL_ARGMAX}] giving the position in the argument list (after the format argument) of an integer argument containing the field width or precision, for example:

```
printf("%1$d:%2$. *3$d:%4$. *3$d\n", hour, min, precision, sec);
```

The format can contain either numbered argument specifications (that is, %n\$ and *m\$), or unnumbered argument specifications (that is, % and *), but normally not both. The only exception to this is that %% can be mixed with the %n\$ form. The results of mixing numbered and unnumbered argument specifications in a format string are undefined. When numbered argument specifications are used, specifying the Nth argument requires that all the leading arguments, from the first to the (N-1)th, are specified in the format string.

The flag characters and their meanings are:

- ' The integer portion of the result of a decimal conversion (%i, %d, %u, %f, %g or %G) will be formatted with thousands' grouping characters. For other conversions the behavior is undefined. The non-monetary grouping character is used.
- The result of the conversion will be left-justified within the field. The conversion will be right-justified if this flag is not specified.
- + The result of a signed conversion will always begin with a sign (+ or -). The conversion 8116 will begin with a sign only when a negative value is converted if this flag is not specified.
- space** If the first character of a signed conversion is not a sign or if a signed conversion results in no characters, a space will be prefixed to the result. This means that if the space and + flags both appear, the space flag will be ignored.
- # This flag specifies that the value is to be converted to an alternative form. For o conversion, it increases the precision (if necessary) to force the first digit of the result to be 0. For x or X conversions, a non-zero result will have 0x (or 0X) prefixed to it. For e, E, f, g or G conversions, the result will always contain a radix character, even if no digits follow the radix character. Without this flag, a radix character appears in the result of these conversions only if a digit follows it. For g and G conversions, trailing zeros will not be removed from the result as they normally are. For other conversions, the behavior is undefined.
- 0** For d, i, o, u, x, X, e, E, f, g and G conversions, leading zeros (following any indication of sign or base) are used to pad to the field width; no space padding is performed. If the 0 and - flags both appear, the 0 flag will be ignored. For d, i, o, u, x and X conversions, if a precision is specified, the 0 flag will be ignored. If the 0 and ' flags both appear, the grouping characters are inserted before zero padding. For other conversions, the behavior is undefined.

The conversion characters and their meanings are:

- d, i** The int argument is converted to a signed decimal in the style [-]dddd. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting 0 with an explicit precision of 0 is no characters.
- o** The unsigned int argument is converted to unsigned octal format in the style dddd. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting 0 with an explicit precision of 0 is no characters.
- u** The unsigned int argument is converted to unsigned decimal format in the style dddd. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting 0 with an explicit precision of 0 is no characters
- x** The unsigned int argument is converted to unsigned hexadecimal format in the style dddd; the letters abcdef are used. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting 0 with an explicit precision of 0 is no characters. /TD>
- X** Behaves the same as the x conversion character except that letters ABCDEF are used instead of abcdef.
- f** The double argument is converted to decimal notation in the style [-]ddd.ddd, where the number of digits after the radix character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is explicitly 0 and no # flag is present, no radix character appears. If a radix character appears, at least one digit appears before it. The value is rounded to the appropriate number of digits. The **fprintf** family of functions may make available character string representations for infinity and NaN.

- e, E** The double argument is converted in the style `[-]d.ddde +/- dd`, where there is one digit before the radix character (which is non-zero if the argument is non-zero) and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is 0 and no # flag is present, no radix character appears. The value is rounded to the appropriate number of digits. The E conversion character will produce a number with E instead of e introducing the exponent. The exponent always contains at least two digits. If the value is 0, the exponent is 0.
- The **fprintf** family of functions may make available character string representations for infinity and NaN.
- g, G** The double argument is converted in the style f or e (or in the style E in the case of a G conversion character), with the precision specifying the number of significant digits. If an explicit precision is 0, it is taken as 1. The style used depends on the value converted; style e (or E) will be used only if the exponent resulting from such a conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional portion of the result; a radix character appears only if it is followed by a digit. The **fprintf** family of functions may make available character string representations for infinity and NaN.
- c** The int argument is converted to an unsigned char, and the resulting byte is written. If an l (ell) qualifier is present, the **wint_t** argument is converted as if by an ls conversion specification with no precision and an argument that points to a two-element array of type **wchar_t**, the first element of which contains the **wint_t** argument to the ls conversion specification and the second element contains a null wide-character.
- s** The argument must be a pointer to an array of char. Bytes from the array are written up to (but not including) any terminating null byte. If the precision is specified, no more than that many bytes are written. If the precision is not specified or is greater than the size of the array, the array must contain a null byte. If an l (ell) qualifier is present, the argument must be a pointer to an array of type **wchar_t**. Wide-characters from the array are converted to characters (each as if by a call to the **wcrtomb** function, with the conversion state described by an **mbstate_t** object initialized to zero before the first wide-character is converted) up to and including a terminating null wide-character. The resulting characters are written up to (but not including) the terminating null character (byte). If no precision is specified, the array must contain a null wide-character. If a precision is specified, no more than that many characters (bytes) are written (including shift sequences, if any), and the array must contain a null wide-character if, to equal the character sequence length given by the precision, the function would need to access a wide-character one past the end of the array. In no case is a partial character written.
- p** The argument must be a pointer to void. The value of the pointer is converted to a sequence of printable characters, in an implementation-dependent manner. n The argument must be a pointer to an integer into which is written the number of bytes written to the output so far by this call to one of the **fprintf** functions. No argument is converted.
- C** Same as lc.
- S** Same as ls.
- %** Print a %; no argument is converted. The entire conversion specification must be %%.

If a conversion specification does not match one of the above forms, the behavior is undefined.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by **fprintf** and **printf** are printed as if **fputc** had been called.

The **st_ctime** and **st_mtime** fields of the file will be marked for update between the call to a successful execution of **fprintf** or **printf** and the next successful completion of a call to **fflush** or **fclose** on the same stream or a call to **exit** or **abort**.

Return Values

Upon successful completion, these functions return the number of bytes transmitted excluding the terminating null in the case of **sprintf** or **snprintf** or a negative value if an output error was encountered.

If the value of **n** is zero on a call to **snprintf**, an unspecified value less than 1 is returned.

Error Codes

For the conditions under which **fprintf** and **printf** will fail and may fail, refer to **fputc** or **fputwc**.

In addition, all forms of **fprintf** may fail if:

EILSEQ A wide-character code that does not correspond to a valid character has been detected.
EINVAL There are insufficient arguments.

In addition, **printf** and **fprintf** may fail if:

ENOMEM Insufficient storage space is available.

Implementation Specifics

If the application calling **fprintf** has any objects of type `wint_t` or `wchar_t`, it must also include the header `wchar.h` to have these objects defined. This subroutine is part of the Base Operating System (BOS) subroutine.

Related Information

The **fputc** subroutine, **fscanf** (“scanf, fscanf, sscanf, or wscanf Subroutine” on page 82) subroutine, **setlocale** (“setlocale Subroutine” on page 107) subroutine, **wctomb** (“wctomb Subroutine” on page 412) subroutine, `wchar.h` file.

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)

struct SpmiHotSet *HotSet;
char *StatName;
SpmiCxHdl GrandParent;
int maxresp;
int threshold;
int frequency;
int feed_type;
int excp_type;
int severity;
int 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.

Parameters

HotSet

Specifies a pointer to a valid structure of type **SpmiHotSet** as created by the **SpmiCreateHotSet** (“SpmiCreateHotSet” on page 173) 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 *excp_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 $2 \times 3 = 6$ 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 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 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.

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-valset for each of **RTime/ARM/checking** and **RTime/ARM/savings**. As new applications are added to the **RTime/ARM** contexts, new pseudo-valsets 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.

Implementation Specifics

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.

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.

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, and the **SpmiErrmsg** variable contains text, in English, explaining the cause of the error. See the *List of SPMI Error Codes* for more information.

Implementation Specifics

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:

- “SpmiDelSetHot Subroutine” on page 179
- “SpmiFreeHotSet Subroutine” on page 187
- “SpmiAddSetHot Subroutine” on page 170
- *Understanding SPMI Data Areas*

SpmiCreateStatSet Subroutine

Purpose

Creates an empty set of statistics.

Library

SPMI Library (`libSpmi.a`)

Syntax

```
#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.

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 **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.

Implementation Specifics

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

<code>/usr/include/sys/Spmidef.h</code>	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 181
- “SpmiFreeStatSet Subroutine” on page 188
- “SpmiPathAddSetStat Subroutine” on page 207
- *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.

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.

Implementation Specifics

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:

- “SpmiDdsDelCx Subroutine”
- “SpmiDdsInit Subroutine” on page 178

SpmiDdsDelCx Subroutine

Purpose

Deletes a volatile context.

Library

SPMI Library (**libSpmi.a**)

Syntax

```
#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.

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.

Implementation Specifics

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:

- “SpmiDdsAddCx Subroutine” on page 175

- “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

```
#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.

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 **SpmiDdslnit** 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 **SpmiDdslnit** 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.

Implementation Specifics

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:

- “SpmiExit Subroutine” on page 182
- “Spmilnit Subroutine” on page 196
- *Understanding SPMI Data Areas*

SpmiDelSetHot Subroutine

Purpose

Removes a single set of peer statistics from a hotset.

Library

SPMI Library (**libSpmi.a**)

Syntax

```
#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.

Parameters

HotSet

Specifies a pointer to a valid structure of type **SpmiHotSet**, as created by the “SpmiCreateHotSet” on page 173 subroutine call.

HotVal

Specifies a pointer to a valid structure of type **SpmiHotVals**, as created by the “SpmiAddSetHot Subroutine” on page 170 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.

Implementation Specifics

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:

- “SpmiCreateHotSet” on page 173
- “SpmiFreeHotSet Subroutine” on page 187
- “SpmiAddSetHot Subroutine” on page 170
- *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
int SpmiDelSetStat(StatSet, StatVal)
struct SpmiStatSet *StatSet;
struct SpmiStatVals *StatVal;
```

Description

The **SpmiDelSetStat** subroutine removes a single statistic, identified by the *StatVal* parameter, from a set of statistics, identified by the *StatSet* parameter.

Parameters

StatSet

Specifies a pointer to a valid structure of type **SpmiStatSet** as created by the “SpmiCreateStatSet Subroutine” on page 174 subroutine call.

StatVal

Specifies a pointer to a valid structure of type **SpmiStatVals** as created by the “SpmiPathAddSetStat Subroutine” on page 207 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 **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.

Implementation Specifics

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:

- “SpmiCreateStatSet Subroutine” on page 174
- “SpmiFreeStatSet Subroutine” on page 188
- “SpmiPathAddSetStat Subroutine” on page 207
- *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

```
#include sys/Spmidef.h
void SpmiExit()
```

Description

A successful “SpmiInit Subroutine” on page 196 or “SpmiDdsInit Subroutine” on page 178 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.

Implementation Specifics

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 196
- “SpmiDdsInit Subroutine” on page 178

SpmiFirstCx Subroutine

Purpose

Locates the first subcontext of a context.

Library

SPMI Library (**libSpmi.a**)

Syntax

```
#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 **SpmiCx** structure through the **SpmiGetCx** subroutine call.

Parameters

CxHandle

Specifies a valid **SpmiCxHdl** handle as obtained by another subroutine call.

Return Values

The **SpmiFirstCx** subroutine returns a pointer to an **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:

- 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.

Implementation Specifics

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:

- “SpmiGetCx Subroutine” on page 189
- “SpmiNextCx Subroutine” on page 199
- *Understanding SPMI Data Areas*
- *Understanding the SPMI Data Hierarchy*

SpmiFirstHot Subroutine

Purpose

Locates the first of the sets of peer statistics belonging to a hotset.

Library

SPMI Library (**libSpmi.a**)

Syntax

```
#include sys/Spmidef.h
struct SpmiHotVals *SpmiFirstHot(HotSet)
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.

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.

Implementation Specifics

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:

- “SpmiCreateHotSet” on page 173
- “SpmiAddSetHot Subroutine” on page 170
- “SpmiNextHot Subroutine” on page 200
- “SpmiNextHotItem Subroutine” on page 201
- *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 192 call.

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.

Implementation Specifics

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:

- “SpmiGetStat Subroutine” on page 192
- “SpmiNextStat Subroutine” on page 203
- *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.

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.

Implementation Specifics

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:

- “SpmiCreateStatSet Subroutine” on page 174
- “SpmiNextVals Subroutine” on page 204
- *Understanding SPMI Data Areas*

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.

Parameters

HotSet

Specifies a pointer to a valid structure of type **SpmiHotSet** as created by the “SpmiCreateHotSet” on page 173 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.

Implementation Specifics

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:

- “SpmiCreateHotSet” on page 173
- “SpmiDelSetHot Subroutine” on page 179
- “SpmiAddSetHot Subroutine” on page 170
- *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.

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.

Implementation Specifics

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:

- “SpmiCreateStatSet Subroutine” on page 174
- “SpmiDelSetStat Subroutine” on page 181
- “SpmiPathAddSetStat Subroutine” on page 207
- *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.

Parameters

CxHandle

Specifies a valid **SpmiCxHdl** handle as obtained by another subroutine call.

Return Values

The **SpmiGetCx** subroutine returns 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:

- 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.

Implementation Specifics

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:

- “SpmiFirstCx Subroutine” on page 183
- “SpmiNextCx Subroutine” on page 199
- *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 **SpmiStatSets** and **SpmiHotSets** 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.

Parameters

HotSet

Specifies a pointer to a valid structure of type **SpmiHotSet** as created by the “SpmiCreateHotSet” on page 173 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.

Implementation Specifics

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:

- “SpmiCreateHotSet” on page 173
- “SpmiAddSetHot Subroutine” on page 170

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.

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.

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.

Implementation Specifics

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:

- “SpmiFirstStat Subroutine” on page 185
- “SpmiNextStat Subroutine” on page 203
- *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

```
#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.

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.

Implementation Specifics

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:

- “SpmiCreateStatSet Subroutine” on page 174

- “SpmiPathAddSetStat Subroutine” on page 207

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.

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.

Implementation Specifics

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:

- “SpmiGetStatSet Subroutine” on page 193
- “SpmiCreateStatSet Subroutine” on page 174
- “SpmiPathAddSetStat Subroutine” on page 207
- *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 Spmilnit (TimeOut)
int TimeOut;
```

Description

The **Spmilnit** 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 **Spmilnit** subroutine call before issuing any other subroutine calls to the SPMI.

Note: The **Spmilnit** subroutine is automatically issued by the **SpmiDdslnit** subroutine call. Successive **Spmilnit** subroutine calls are ignored.

Note: If the calling program uses shared memory for other purposes, including memory mapping of files, the **Spmilnit** subroutine call must be issued before access is established to other shared memory areas.

The SPMI entry point called by the **Spmilnit** 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.

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 **Spmilnit** 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 **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.

Implementation Specifics

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:

- “SpmiDdslnit Subroutine” on page 178
- “SpmiExit Subroutine” on page 182

Spmilstantiate Subroutine

Purpose

Explicitly instantiates the subcontexts of an instantiable context.

Library

SPMI Library (**libSpmi.a**)

Syntax

```
#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.

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.

Implementation Specifics

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:

- “SpmiFirstCx Subroutine” on page 183
- “SpmiPathGetCx Subroutine” on page 209
- *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 )struct SpmiCxLink *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.

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.

Implementation Specifics

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:

- “SpmiFirstCx Subroutine” on page 183
- “SpmiGetCx Subroutine” on page 189
- *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 **SpmiHotItems** elements. Use of this subroutine is only necessary if it is desired to skip over some **SpmiHotVals** 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**.

Parameters

HotSet

Specifies a valid pointer to an **SpmiHotSet** structure as obtained by a previous “SpmiCreateHotSet” on page 173 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.

Implementation Specifics

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

<code>/usr/include/sys/Spmidef.h</code>	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 184
- “SpmiGetHotSet Subroutine” on page 191
- “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

```
#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*.

Parameters

HotSet

Specifies a valid pointer to an **SpmiHotSet** structure as obtained by a previous “SpmiCreateHotSet” on page 173 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.

Implementation Specifics

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 more information, see:

- “SpmiFirstHot Subroutine” on page 184
- “SpmiNextHot Subroutine” on page 200
- “SpmiGetHotSet Subroutine” on page 191
- *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 192 call.

Parameters

StatLink

Specifies a valid pointer to a **SpmiStatLink** structure as obtained by a previous “SpmiFirstStat Subroutine” on page 185 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.

Implementation Specifics

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:

- “SpmiFirstStat Subroutine” on page 185
- “SpmiGetStat Subroutine” on page 192
- *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 174 call.

StatVal

Specifies a pointer to a valid structure of type **SpmiStatVals** as created by the “SpmiPathAddSetStat Subroutine” on page 207 subroutine call or returned by a previous “SpmiFirstVals Subroutine” on page 186 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
struct SpmiStatVals*SpmiNextValue( StatSet, StatVal, value)
struct SpmiStatSet *StatSet;
struct SpmiStatVals *StatVal;
float *value;

```

Description

Instead of issuing subroutine calls to “SpmiFirstVals Subroutine” on page 186 / “SpmiNextVals Subroutine” on page 204 (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 193 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 174 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 207 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.

valueA 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.

not NULL

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.

Implementation Specifics

- 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 SPML.

Related Information

For related information, see:

- “SpmiGetStatSet Subroutine” on page 193
- “SpmiCreateStatSet Subroutine” on page 174
- “SpmiPathAddSetStat Subroutine”.
- *Data Access Structures and Handles, StatSets*

SpmiPathAddSetStat Subroutine

Purpose

Adds a statistics value to a set of statistics.

Library

SPML 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.

Parameters

StatSet

Specifies a pointer to a valid structure of type **SpmiStatSet** as created by the “SpmiCreateStatSet Subroutine” on page 174 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.

Implementation Specifics

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:

- “SpmiGetStatSet Subroutine” on page 193
- “SpmiCreateStatSet Subroutine” on page 174
- “SpmiDelSetStat Subroutine” on page 181
- “SpmiFreeStatSet Subroutine” on page 188.
- *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.

Parameters

CxPath

Specifies 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.

Implementation Specifics

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:

- *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

```
#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.

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.

Implementation Specifics

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:

- *Understanding SPMI Data Areas*
- *Understanding the SPMI Data Hierarchy*

sqrt, sqrtl, or cbrt Subroutine

Purpose

Computes square root and cube root functions.

Libraries

IEEE Math Library (**libm.a**)

or System V Math Library (**libmsaa.a**)

Syntax

```
#include <math.h>
double sqrt ( x)
double x;

long double sqrtl (x)
long double x;

double cbrt (x)
double x;
```

Description

The **sqrt** subroutine, **sqrtl** subroutine, and **cbrt** subroutine compute the square root and cube root, respectively, of their parameters.

Note: Compile any routine that uses subroutines from the **libm.a** library with the **-lm** flag. To compile the **sqrt.c** file, for example, enter:

```
cc sqrt.c -lm
```

Parameters

x Specifies some double-precision floating-point value.

Return Values

The **sqrt** (-0.0) = -0.0.

The **sqrtl** (-0.0) = -0.0.

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):

For the **sqrt** and the **sqrtl** subroutines, 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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

The **cbrt** subroutine is not part of the ANSI C Library.

Related Information

The **exp**, **expm1**, **log**, **log10**, **log1p**, or **pow** subroutine.

Subroutines Overview *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs.*

128-Bit long double Floating-Point Format *AIX 5L Version 5.1 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 (**libsrc.a**)

Syntax

```
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

<i>errno</i>	Specifies the SRC error code.
<i>ErrorText</i>	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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **addssys** subroutine, **chssys** subroutine, **delssys** subroutine, **defssys** subroutine, **getsubsvr** subroutine, **getssys** subroutine, **srcsbuf** (“srcsbuf Subroutine” on page 217) subroutine, **srcrrqs** (“srcrrqs Subroutine” on page 214) subroutine, **srcsrpy** (“srcsrpy Subroutine” on page 223) subroutine, **srcsrqt** (“srcsrqt Subroutine” on page 226) subroutine, **srcstat** (“srcstat Subroutine” on page 233) subroutine, **srcstathdr** (“srcstathdr Subroutine” on page 237) subroutine, **srcstattxt** (“srcstattxt Subroutine” on page 238) subroutine, **srcstop** (“srcstop Subroutine” on page 239) subroutine, **srcstrt** (“srcstrt Subroutine” on page 242) subroutine.

List of SRC Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Programming Subsystem Communication with the SRC in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

System Resource Controller (SRC) Overview for Programmers in *AIX 5L Version 5.1 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

<i>srcerrno</i>	Specifies the SRC error code.
<i>ErrorText</i>	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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **srcsbuf_r** (“srcsbuf_r Subroutine” on page 220), **srcsrqt_r** (“srcsrqt_r Subroutine” on page 229), **srcrrqs_r** (“srcrrqs_r Subroutine” on page 216), **srcstat_r** (“srcstat_r Subroutine” on page 235), and **srcstattxt_r** (“srcstattxt_r Subroutine” on page 239) subroutines.

List of SRC Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Programming Subsystem Communication with the SRC in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

System Resource Controller (SRC) Overview for Programmers in *AIX 5L Version 5.1 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:

```
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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Files

/dev/SRC	Specifies the AF_UNIX socket file.
/dev/.SRC-unix	Specifies the location for temporary socket files.

Related Information

The **srcsbuf** (“srcsbuf Subroutine” on page 217) subroutine, **srcsrpy** (“srcsrpy Subroutine” on page 223) subroutine, **srcsrqt** (“srcsrqt Subroutine” on page 226) subroutine, **srcstat** (“srcstat Subroutine” on page 233) subroutine, **srcstathdr** (“srcstathdr Subroutine” on page 237) subroutine, **srcstattxt** (“srcstattxt Subroutine” on page 238) subroutine, **srcstop** (“srcstop Subroutine” on page 239) subroutine, **srcstrt** (“srcstrt Subroutine” on page 242) subroutine.

List of SRC Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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

```
#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:

```
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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **src_err_msg_r** (“src_err_msg_r Subroutine” on page 213), **srcsbuf_r** (“srcsbuf_r Subroutine” on page 220), **srcsrqt_r** (“srcsrqt_r Subroutine” on page 229), **srcstat_r** (“srcstat_r Subroutine” on page 235), and **srcstattxt_r** (“srcstattxt_r Subroutine” on page 239) subroutines.

srcsbuf 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>
```

```
intsrcsbuf(Host,Type,SubsystemName,
SubserverObject,SubsystemPID, StatusType,StatusFrom,StatusText,Continued)
```

```
char * Host, * SubsystemName;
```

```
char * SubserverObject, ** StatusText;
```

```
short Type, StatusType;
```

```
int SubsystemPID, StatusFrom, * Continued;
```

Description

The **srcsbuf** 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 **srcrsqt** subroutine is called to get the long status of one subsystem. When the *Type* parameter is not **SUBSYSTEM**, the **srcrsqt** subroutine is called to get the long or short status of a subserver.

Parameters

<i>Host</i>	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.
<i>Type</i>	Specifies whether the status request applies to the subsystem or subserver. If the <i>Type</i> parameter is set to SUBSYSTEM , the status request is for a subsystem. If not, the status request is for a subserver and the <i>Type</i> parameter is a subserver code point.
<i>SubsystemName</i>	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 <i>SubsystemName</i> 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 <i>SubsystemName</i> parameter, you must specify a <i>SubsystemPID</i> parameter.
<i>SubserverObject</i>	Specifies a subserver object. The <i>SubserverObject</i> parameter modifies the <i>Type</i> parameter. The <i>SubserverObject</i> parameter is ignored if the <i>Type</i> parameter is set to SUBSYSTEM . The use of the <i>SubserverObject</i> parameter is determined by the subsystem and the caller. This parameter will be placed in the <i>objname</i> field of the subreq structure that is passed to the subsystem.
<i>SubsystemPID</i>	Specifies the process ID of the subsystem on which to get status, as returned by the srcstr subroutine. You must specify the <i>SubsystemPID</i> parameter if multiple instances of the subsystem are active and you request a long subsystem status or subserver status. If you specify a null <i>SubsystemPID</i> parameter, you must specify a <i>SubsystemName</i> parameter.
<i>StatusType</i>	Specifies LONGSTAT for long status or SHORTSTAT for short status.
<i>StatusFrom</i>	Specifies whether status errors and messages are to be printed to standard output or just returned to the caller. When the <i>StatusFrom</i> parameter is SSHELL , the errors are printed to standard output.
<i>StatusText</i>	Allocates memory for the printable text and sets the <i>StatusText</i> parameter to point to this memory. After it prints the text, the calling process must free the memory allocated for this buffer.
<i>Continued</i>	Specifies whether this call to the srcsbuf subroutine is a continuation of a status request. If the <i>Continued</i> 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 <i>Continued</i> parameter is set to END or STATCONTINUED by the subsystem. If the <i>Continued</i> 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 <i>Continued</i> parameter to a value other than NEWREQUEST . The <i>Continued</i> 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.
--------------------	--

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 <i>/etc/hosts.equiv</i> 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_MMRV	An SRC component could not allocate the memory it needs.
SRC_NOCONTINUE	The <i>Continued</i> 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 <i>/etc/services</i> 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:

```
char *status;
int continued=NEWREQUEST;
int rc;

do {
    rc=srscbuf("MaryC", SUBSYSTEM, "srctest", "", 0,
        SHORTSTAT, SSHELL, &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=srscbuf("", 12345, "srctest", "", 0,
        LONGSTAT, SSHELL, &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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Files

<code>/etc/services</code>	Defines sockets and protocols used for Internet services.
<code>/dev/SRC</code>	Specifies the AF_UNIX socket file.
<code>/dev.SRC-unix</code>	Specifies the location for temporary socket files.

Related Information

The **srcrrqs** (“srcrrqs Subroutine” on page 214) subroutine, **srcsrpy** (“srcsrpy Subroutine” on page 223) subroutine, **srcsrqt** (“srcsrqt Subroutine” on page 226) subroutine, **srcstat** (“srcstat Subroutine” on page 233) subroutine, **srcstathdr** (“srcstathdr Subroutine” on page 237) subroutine, **srcstattxt** (“srcstattxt Subroutine” on page 238) subroutine, **srcstop** (“srcstop Subroutine” on page 239) subroutine, **srcstrt** (“srcstrt Subroutine” on page 242) subroutine.

List of SRC Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Programming Subsystem Communication with the SRC in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

System Resource Controller (SRC) Overview for Programmers in *AIX 5L Version 5.1 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 **srcstat_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

<i>Host</i>	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.
<i>Type</i>	Specifies whether the status request applies to the subsystem or subserver. If the <i>Type</i> parameter is set to SUBSYSTEM , the status request is for a subsystem. If not, the status request is for a subserver and the <i>Type</i> parameter is a subserver code point.
<i>SubsystemName</i>	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 <i>SubsystemName</i> 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 <i>SubsystemName</i> parameter, you must specify a <i>SubsystemPID</i> parameter.
<i>SubserverObject</i>	Specifies a subserver object. The <i>SubserverObject</i> parameter modifies the <i>Type</i> parameter. The <i>SubserverObject</i> parameter is ignored if the <i>Type</i> parameter is set to SUBSYSTEM . The use of the <i>SubserverObject</i> 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.
<i>SubsystemPID</i>	Specifies the process ID of the subsystem on which to get status, as returned by the srcstrt subroutine. You must specify the <i>SubsystemPID</i> parameter if multiple instances of the subsystem are active and you request a long subsystem status or subserver status. If you specify a null <i>SubsystemPID</i> parameter, you must specify a <i>SubsystemName</i> parameter.
<i>StatusType</i>	Specifies LONGSTAT for long status or SHORTSTAT for short status.
<i>StatusFrom</i>	Specifies whether status errors and messages are to be printed to standard output or just returned to the caller. When the <i>StatusFrom</i> 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.
<i>StatusText</i>	Allocates memory for the printable text and sets the <i>StatusText</i> parameter to point to this memory. After it prints the text, the calling process must free the memory allocated for this buffer.
<i>Continued</i>	Specifies whether this call to the srcsbuf_r subroutine is a continuation of a status request. If the <i>Continued</i> 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 <i>Continued</i> 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 <i>Continued</i> 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 <i>Continued</i> parameter to a value other than NEWREQUEST . In normal processing, the <i>Continued</i> 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.
<i>SRCHandle</i>	Identifies a request and its associated responses. Set to NULL by the caller for a NEWREQUEST . The srcsbuf_r subroutine saves a value in <i>SRCHandle</i> to allow srcsbuf_r continuation calls to use the same socket and internal buffers. The <i>SRCHandle</i> parameter should not be changed by the caller except for NEWREQUEST s.

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:

<code>SRC_BADSOCK</code>	The request could not be passed to the subsystem because of some socket failure.
<code>SRC_CONT</code>	The subsystem uses signals. The request cannot complete.
<code>SRC_DMNA</code>	The SRC daemon is not active.
<code>SRC_INET_AUTHORIZED_HOST</code>	The local host is not in the remote <code>/etc/hosts.equiv</code> file.
<code>SRC_INET_INVALID_HOST</code>	On the remote host, the local host is not known.
<code>SRC_INVALID_USER</code>	The user is not root or group system.
<code>SRC_MMRV</code>	An SRC component could not allocate the memory it needs.
<code>SRC_NOCONTINUE</code>	The <code>Continued</code> parameter was not set to <code>NEWREQUEST</code> , and no continuation is currently active.
<code>SRC_NORPLY</code>	The request timed out waiting for a response.
<code>SRC_NSVR</code>	The subsystem is not active.
<code>SRC_SOCKET</code>	There is a problem with SRC socket communications.
<code>SRC_STPG</code>	The request was not passed to the subsystem. The subsystem is stopping.
<code>SRC_UDP</code>	The SRC port is not defined in the <code>/etc/services</code> file.
<code>SRC_UHOST</code>	The foreign host is not known.
<code>SRC_WICH</code>	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;
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:

```

char *status;
int continued=NEWREQUEST;
int rc;
char *handle

do {
    rc=srcsbuf_r("", 12345, "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 srcsbuf_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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **src_err_msg_r** ("src_err_msg_r Subroutine" on page 213) subroutine, **srcsrqt_r** ("srcsrqt_r Subroutine" on page 229) subroutine, **srcrrqs_r** ("srcrrqs_r Subroutine" on page 216) subroutine, **srcstat_r** ("srcstat_r Subroutine" on page 235) subroutine, **srcstattxt_r** ("srcstattxt_r Subroutine" on page 239) subroutine.

List of SRC Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Programming Subsystem Communication with the SRC in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

System Resource Controller (SRC) Overview for Programmers in *AIX 5L Version 5.1 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

<i>SRChdr</i>	Points to the reply address buffer as returned by the srcrrqs subroutine.
<i>PPacket</i>	Points to the reply packet. The first element of the reply packet is a srchdr structure. The cont element of the <i>PPacket</i> -> srchdr structure is modified on returning from the srcsrpy subroutine. The second element of the reply packet should be a svrreply structure, an array of statcode structures, or another format upon which the subsystem and the requester have agreed.
<i>PPacketSize</i>	Specifies the number of bytes in the reply packet pointed to by the <i>PPacket</i> parameter. The <i>PPacketSize</i> 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.
<i>Continued</i>	Indicates whether this reply is to be continued. If the <i>Continued</i> parameter is set to the constant END , no more reply packets are sent for this request. If the <i>Continued</i> parameter is set to CONTINUED , the second element of what is indicated by the <i>PPacket</i> parameter must be a svrreply structure, since the <i>rtmsg</i> element of the svrreply structure is printed to standard output. For a status reply, the <i>Continued</i> parameter is set to STATCONTINUED , and the second element of what is pointed to by the <i>PPacket</i> parameter must be an array of statcode structures. If a STOP subsystem request is received, only one reply packet can be sent and the <i>Continued</i> 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 <i>Continued</i> parameter. Values other than the following must be used:
0	END
1	CONTINUED
2	STATCONTINUED

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:

SRC SOCK	There is a problem with SRC socket communications.
SRC_REPLYSZ	SRC reply size is invalid.

Examples

1. To send a **STOP** subsystem reply, enter:

```
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");

srcsrpy(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) */
srcsrpy(srchdr,&status,sizeof(status),STATCONTINUED);
.
.
srcsrpy(srchdr,&status,sizeof(status),STATCONTINUED);

/* send final packet */
srcsrpy(srchdr,&status,sizeof(struct srchdr),END);
```

This entry sends several status packets.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Files

`/dev.SRC-unix` Specifies the location for temporary socket files.

Related Information

The **srcrrqs** (“srcrrqs Subroutine” on page 214) subroutine, **srcsbuf** (“srcsbuf Subroutine” on page 217) subroutine, **srcsrqt** (“srcsrqt Subroutine”) subroutine, **srcstat** (“srcstat Subroutine” on page 233) subroutine, **srcstathdr** (“srcstathdr Subroutine” on page 237) subroutine, **srcstattxt** (“srcstattxt Subroutine” on page 238) subroutine, **srcstop** (“srcstop Subroutine” on page 239) subroutine, **srcstrt** (“srcstrt Subroutine” on page 242) subroutine.

List of SRC Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Programming Subsystem Communication with the SRC in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

System Resource Controller (SRC) Overview for Programmers in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Understanding SRC Communication Types in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

srcsrqt Subroutine

Purpose

Sends a request to a subsystem.

Library

System Resource Controller Library (**libsrc.a**)

Syntax

```
#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	<i>ReplyBuffer->srchdr.continued</i> is set to the END constant.
Reply continuation	<i>ReplyBuffer->srchdr.continued</i> 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

<i>SubsystemPID</i>	The process ID of the subsystem.
<i>Host</i>	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/inittab) must be started with the -r flag and the /etc/hosts.equiv or .rhosts file must be configured to allow remote requests.
<i>SubsystemName</i>	Specifies the name of the subsystem to which this request is to be sent. You must specify a <i>SubsystemName</i> if you do not specify a <i>SubsystemPID</i> .
<i>RequestLength</i>	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.
<i>SubsystemRequest</i>	Points to the subsystem request packet.
<i>ReplyLength</i>	Specifies the maximum length, in bytes, of the reply to be received from the subsystem. On return from the srcsrqt subroutine, the <i>ReplyLength</i> parameter is set to the actual length of the subsystem reply packet.
<i>ReplyBuffer</i>	Points to a buffer for the receipt of the reply packet from the subsystem.
<i>StartItAlso</i>	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.
<i>Continued</i>	Specifies whether this call to the srcsrqt subroutine is a continuation of a previous request. If the <i>Continued</i> 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 <i>Continued</i> to any value other than NEWREQUEST . The last response from the subsystem will set <i>Continued</i> 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:

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_MMRV	An SRC component could not allocate the memory it needs.
SRC_NOCONTINUE	The <i>Continued</i> 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
SRC_SOCK
SRC_STPG

The *RequestLength* is greater than the maximum 2000 bytes.
There is a problem with SRC socket communications.
The request was not passed to the subsystem. The subsystem is stopping.

SRC_UDP
SRC_UHOST

The SRC port is not defined in the */etc/services* file.
The foreign host is not known.

Examples

1. To request long subsystem status, enter:

```
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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Files

<code>/etc/services</code>	Defines sockets and protocols used for Internet services.
<code>/dev/SRC</code>	Specifies the AF_UNIX socket file.
<code>/dev/.SRC-unix</code>	Specifies the location for temporary socket files.

Related Information

The **srcrrqs** (“srcrrqs Subroutine” on page 214) subroutine, **srcsbuf** (“srcsbuf Subroutine” on page 217) subroutine, **srcsrpy** (“srcsrpy Subroutine” on page 223) subroutine, **srcstat** (“srcstat Subroutine” on page 233) subroutine, **srcstathdr** (“srcstathdr Subroutine” on page 237) subroutine, **srcstattxt** (“srcstattxt Subroutine” on page 238) subroutine, **srcstop** (“srcstop Subroutine” on page 239) subroutine, **srcstrt** (“srcstrt Subroutine” on page 242) subroutine.

List of SRC Subroutines, Programming Subsystem Communication with the SRC, System Resource Controller (SRC) Overview for Programmers in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

srcsrqt_r Subroutine

Purpose

Sends a request to a subsystem.

Library

System Resource Controller Library (**libsrc.a**)

Syntax

```
#include <spc.h>
```

```

srcsrqt_r(Host, SubsystemName, SubsystemPID, RequestLength,
          SubsystemRequest, ReplyLength, ReplyBuffer, StartItAlso,
          Continued, SRCHandle)
char * Host, * SubsystemName;
char * SubsystemRequest, * ReplyBuffer;
pid_t SubsystemPID,
int, StartItAlso, * Continued;
short RequestLength, * ReplyLength;
char ** SRCHandle;

```

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	<i>ReplyBuffer->srchdr.continued</i> is set to the END constant.
Reply continuation	<i>ReplyBuffer->srchdr.continued</i> 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

<i>SubsystemPID</i>	The process ID of the subsystem.
<i>Host</i>	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.
<i>SubsystemName</i>	Specifies the name of the subsystem to which this request is to be sent. You must specify a <i>SubsystemName</i> if you do not specify a <i>SubsystemPID</i> .
<i>RequestLength</i>	Specifies the length, in bytes, of the request to be sent to the subsystem. The maximum length is 2000 bytes.
<i>SubsystemRequest</i>	Points to the subsystem request packet.
<i>ReplyLength</i>	Specifies the maximum length, in bytes, of the reply to be received from the subsystem. On return from the srcsrqt subroutine, the <i>ReplyLength</i> parameter is set to the actual length of the subsystem reply packet.
<i>ReplyBuffer</i>	Points to a buffer for the receipt of the reply packet from the subsystem.
<i>StartItAlso</i>	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.
<i>Continued</i>	Specifies whether this call to the srcsrqt subroutine is a continuation of a previous request. If the <i>Continued</i> 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 <i>Continued</i> to any value other than NEWREQUEST . The last response from the subsystem will set <i>Continued</i> 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 <i>Continued</i> = SRC_CLOSE to discard the remaining data, close the socket, and free the internal buffers.
<i>SRCHandle</i>	Identifies a request and its associated responses. Set to NULL by the caller for a NEWREQUEST . The srcsrqt_r subroutine saves a value in <i>SRCHandle</i> to allow srcsrqt_r continuation calls to use the same socket and internal buffers. The <i>SRCHandle</i> 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 <code>/etc/hosts.equiv</code> 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_MMRV	An SRC component could not allocate the memory it needs.
SRC_NOCONTINUE	The <i>Continued</i> 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 <i>RequestLength</i> is greater than the maximum 2000 bytes.
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 <code>/etc/services</code> 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

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** (“src_err_msg_r Subroutine” on page 213), **srcsbuf_r** (“srcsbuf_r Subroutine” on page 220), **srcrrqs_r** (“srcrrqs_r Subroutine” on page 216), **srcstat_r** (“srcstat_r Subroutine” on page 235), and **srcstattxt_r** (“srcstattxt_r Subroutine” on page 239) subroutines.

List of SRC Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Programming Subsystem Communication with the SRC in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

System Resource Controller (SRC) Overview for Programmers in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

srcstat Subroutine

Purpose

Gets short status on a subsystem.

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;  
struct statrep * 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

<i>Host</i>	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.
<i>SubsystemName</i>	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 <i>SubsystemName</i> 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 <i>SubsystemName</i> parameter, you must specify a <i>SubsystemPID</i> parameter.
<i>SubsystemPID</i>	Specifies the PID of the subsystem on which to get status as returned by the srcstat subroutine. You must specify the <i>SubsystemPID</i> parameter if multiple instances of the subsystem are active and you request a long subsystem status or subserver status. If you specify a null <i>SubsystemPID</i> parameter, you must specify a <i>SubsystemName</i> parameter.
<i>ReplyLength</i>	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.
<i>StatusReply</i>	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.
<i>Continued</i>	Specifies whether this call to the srcstat subroutine is a continuation of a previous status request. If the <i>Continued</i> parameter is set to NEWREQUEST , 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 <i>Continued</i> to a value other than NEWREQUEST . The last response for the SRC sets <i>Continued</i> 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_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_MMRV	An SRC component could not allocate the memory it needs.
SRC_NOCONTINUE	<i>Continued</i> was not set to NEWREQUEST and no continuation is currently active.
SRC_NORPLY	The request timed out waiting for a response.
SRC_SOCKET	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:

```
intcont=NEWREQUEST;
struct statcode statcode[20];
short replen=sizeof(statcode);

srcstat("MaryC","srctest",0,&replen,statcode,&cont);
```

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:

```
intcont=NEWREQUEST;
struct statcode statcode[20];
short replen=sizeof(statcode);

srcstat("",SRCALLSUBSYS,0,&replen,statcode,&cont);
```

This entry requests short status of all subsystems on the local machine.

3. To request the status for a group of subsystems, enter:

```
intcont=NEWREQUEST;
struct statcode statcode[20];
short replen=sizeof(statcode);
char subsysname[30];

strcpy(subsysname,SRCGROUP);
strcat(subsysname,"tcpip");
srcstat("",subsysname,0,&replen,statcode, &cont);
```

This entry requests short status of all members of the subsystem group **tcpip** on the local machine.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

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** (“srcrrqs Subroutine” on page 214) subroutine, **srcsbuf** (“srcsbuf Subroutine” on page 217) subroutine, **srcsrpy** (“srcsrpy Subroutine” on page 223) subroutine, **srcsrqt** (“srcsrqt Subroutine” on page 226) subroutine, **srcstathdr** (“srcstathdr Subroutine” on page 237) subroutine, **srcstattxt** (“srcstattxt Subroutine” on page 238) subroutine, **srcstop** (“srcstop Subroutine” on page 239) subroutine, **srcstrt** (“srcstrt Subroutine” on page 242) subroutine.

List of SRC Subroutines, Programming Subsystem Communication with the SRC, System Resource Controller (SRC) Overview for Programmers in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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

<i>Host</i>	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.
<i>SubsystemName</i>	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 <i>SubsystemName</i> 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 <i>SubsystemName</i> parameter, you must specify a <i>SubsystemPID</i> parameter.
<i>SubsystemPID</i>	Specifies the PID of the subsystem on which to get status as returned by the srcstat_r subroutine. You must specify the <i>SubsystemPID</i> parameter if multiple instances of the subsystem are active and you request a long subsystem status or subserver status. If you specify a null <i>SubsystemPID</i> parameter, you must specify a <i>SubsystemName</i> parameter.
<i>ReplyLength</i>	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.
<i>StatusReply</i>	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, allocates 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.

SRCHandle

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 <i>/etc/hosts.equiv</i> 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_MMRV	An SRC component could not allocate the memory it needs.
SRC_NOCONTINUE	<i>Continued</i> was not set to NEWREQUEST and no continuation is currently active.
SRC_NORPLY	The request timed out waiting for a response.
SRC_SOCKET	There is a problem with SRC socket communications.
SRC_UDP	The SRC port is not defined in the <i>/etc/services</i> file.
SRC_UHOST	The foreign host is not known.

Examples

1. To request the status of a subsystem, enter:

```
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:

```
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:

```
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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Files

<code>/etc/services</code>	Defines the sockets and protocols used for Internet services.
<code>/dev/SRC</code>	Specifies the AF_UNIX socket file.
<code>/dev.SRC-unix</code>	Specifies the location for temporary socket files.

Related Information

The `src_err_msg_r` (“src_err_msg_r Subroutine” on page 213), `srcsbuf_r` (“srcsbuf_r Subroutine” on page 220), `srcsrqt_r` (“srcsrqt_r Subroutine” on page 229), `srcrrqs_r` (“srcrrqs_r Subroutine” on page 216), and `srcstattxt_r` (“srcstattxt_r Subroutine” on page 239) subroutines.

srcstathdr Subroutine

Purpose

Gets the title line of the System Resource Controller (SRC) status text.

Library

System Resource Controller Library (**libsrc.a**)

Syntax

```
void srcstathdr ( Title1, Title2)
char *Title1, *Title2;
```

Description

The `srcstathdr` subroutine retrieves the title line, or header, of the SRC status text.

Parameters

<i>Title1</i>	Specifies the objname field of a statcode structure. The subsystem name title is placed here.
<i>Title2</i>	Specifies the objtext field of a statcode structure. The remaining titles are placed here.

Return Values

The subsystem name title is returned in the *Title1* parameter. The remaining titles are returned in the *Title2* parameter.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **srcrrqs** (“srcrrqs Subroutine” on page 214) subroutine, **srcsbuf** (“srcsbuf Subroutine” on page 217) subroutine, **srcsrpy** (“srcsrpy Subroutine” on page 223) subroutine, **srcsrqt** (“srcsrqt Subroutine” on page 226) subroutine, **srcstat** (“srcstat Subroutine” on page 233) subroutine, **srcstattxt** (“srcstattxt Subroutine”) subroutine, **srcstop** (“srcstop Subroutine” on page 239) subroutine, **srcstrt** (“srcstrt Subroutine” on page 242) subroutine.

List of SRC Subroutines, Programming Subsystem Communication with the SRC, System Resource Controller (SRC) Overview for Programmers in *AIX 5L Version 5.1 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **srcrrqs** (“srcrrqs Subroutine” on page 214) subroutine, **srcsbuf** (“srcsbuf Subroutine” on page 217) subroutine, **srcsrpy** (“srcsrpy Subroutine” on page 223) subroutine, **srcsrqt** (“srcsrqt Subroutine” on page 226) subroutine, **srcstat** (“srcstat Subroutine” on page 233) subroutine, **srcstathdr** (“srcstathdr Subroutine” on page 237) subroutine, **srcstop** (“srcstop Subroutine” on page 239) subroutine, **srcstrt** (“srcstrt Subroutine” on page 242) subroutine.

List of SRC Subroutines, Programming Subsystem Communication with the SRC, System Resource Controller (SRC) Overview for Programmers in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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

<i>StatusCode</i>	Specifies an SRC status code to be translated into meaningful text.
<i>Text</i>	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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **src_err_msg_r** (“src_err_msg_r Subroutine” on page 213), **srcsbuf_r** (“srcsbuf_r Subroutine” on page 220), **srcsrqt_r** (“srcsrqt_r Subroutine” on page 229), **srcrrqs_r** (“srcrrqs_r Subroutine” on page 216), and **srcstat_r** (“srcstat_r Subroutine” on page 235) subroutines.

List of SRC Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Programming Subsystem Communication with the SRC in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

System Resource Controller (SRC) Overview for Programmers in *AIX 5L Version 5.1 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

<i>Host</i>	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.
<i>SubsystemName</i>	Specifies the name of the subsystem to stop.
<i>SubsystemPID</i>	Specifies the process ID of the system to stop as returned by the srcstrt subroutine. If you specify a null <i>SubsystemPID</i> parameter, you must specify a <i>SubsystemName</i> parameter.
<i>StopType</i>	Specifies the type of stop requested of the subsystem. If this parameter is null, a normal stop is assumed. The <i>StopType</i> 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.
<i>ReplyLength</i>	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.
<i>ServerReply</i>	Points to an svrreply structure that will receive the subsystem stop reply.
<i>StopFrom</i>	Specifies whether the srcstop subroutine is to display stop results to standard output. If the <i>StopFrom</i> parameter is set to SSHELL , the stop results are displayed to standard output and the srcstop subroutine returns successfully. If the <i>StopFrom</i> 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_MMRV	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_SOCKET	There is a problem with SRC socket communications.
SRC_STPG	The request was not passed to the subsystem. The subsystem is stopping.
SRC_SVND	The subsystem is unknown to the SRC daemon.
SRC_UDP	The remote SRC port is not defined in the /etc/services file.
SRC_UHOST	The foreign host is not known.
SRC_PARM	Invalid parameter passed.

Examples

1. To stop all instances of a subsystem, enter:

```
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:

```
struct svrreply svrreply;
short replen=sizeof(svrreply);

rc=srcstop("", "", 999,CANCEL,&replen,&svrreply,SSHLL);
```

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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

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 214) subroutine, **srcsbuf** (“srcsbuf Subroutine” on page 217) subroutine, **srcsrpy** (“srcsrpy Subroutine” on page 223) subroutine, **srcsrqt** (“srcsrqt Subroutine” on page 226) subroutine, **srcstat** (“srcstat Subroutine” on page 233) subroutine, **srcstathdr** (“srcstathdr Subroutine” on page 237) subroutine, **srcstattxt** (“srcstattxt Subroutine” on page 238) subroutine, **srcstrt** (“srcstrt Subroutine”) subroutine.

List of SRC Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Programming Subsystem Communication with the SRC in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

System Resource Controller (SRC) Overview for Programmers in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

srcstrt Subroutine

Purpose

Starts a System Resource Controller (SRC) subsystem.

Library

System Resource Controller Library (**libsrc.a**)

Syntax

```
#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

<i>Host</i>	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.
<i>SubsystemName</i>	Specifies the name of the subsystem to start.

<i>Environment</i>	Specifies a string that is placed in the subsystem environment when the subsystem is executed. The combined values of the <i>Environment</i> and <i>Arguments</i> 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 <i>Environment</i> value, and blanks outside a quoted string delimit each environment value.
<i>Arguments</i>	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 <i>Environment</i> and <i>Arguments</i> 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.
<i>Restart</i>	Specifies override on subsystem restart. If the <i>Restart</i> parameter is set to SRCNO , the subsystem's restart definition from the subsystem object class is used. If the <i>Restart</i> parameter is set to SRCYES , the restart of a subsystem is not attempted if it terminates abnormally.
<i>StartFrom</i>	Specifies whether the srcstrt subroutine is to display start results to standard output. If the <i>StartFrom</i> parameter is set to SSHHELL , the start results are displayed to standard output, and the srcstrt subroutine always returns successfully. If the <i>StartFrom</i> 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 **SSHHELL**, 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.
SRC_DMNA	The SRC daemon is not active.
SRC_FEXE	The subsystem could not be forked and execed .
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_INPT	The subsystem standard input file could not be established.
SRC_MMRV	An SRC component could not allocate the memory it needs.
SRC_MSGQ	The subsystem message queue could not be created.
SRC_MULT	Multiple instance of the subsystem are not allowed.
SRC_NORPLY	The request timed out waiting for a response.
SRC_OUT	The subsystem standard output file could not be established.
SRC_PIPE	A pipe could not be established for the subsystem.
SRC_SERR	The subsystem standard error file could not be established.
SRC_SUBSOCK	The subsystem communication socket could not be created.
SRC_SUBSYSID	The system user ID is invalid.
SRC_SOCK	There is a problem with SRC socket communications.
SRC_SVND	The subsystem is unknown to the SRC daemon.
SRC_UDP	The SRC port is not defined in the /etc/services header file.
SRC_UHOST	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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Files

<code>/etc/services</code>	Defines sockets and protocols used for Internet services.
<code>/dev/SRC</code>	Specifies the AF_UNIX socket file.
<code>/dev.SRC-unix</code>	Specifies the location for temporary socket files.

Related Information

The **srcrrqs** (“srcrrqs Subroutine” on page 214) subroutine, **srcsbuf** (“srcsbuf Subroutine” on page 217) subroutine, **srcsrpy** (“srcsrpy Subroutine” on page 223) subroutine, **srcsrqt** (“srcsrqt Subroutine” on page 226) subroutine, **srcstat** (“srcstat Subroutine” on page 233) subroutine, **srcstathdr** (“srcstathdr Subroutine” on page 237) subroutine, **srcstattxt** (“srcstattxt Subroutine” on page 238) subroutine, **srcstop** (“srcstop Subroutine” on page 239) subroutine.

List of SRC Subroutines, Programming Subsystem Communication with the SRC, System Resource Controller (SRC) Overview for Programmers in *AIX 5L Version 5.1 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

<i>Signal</i>	Specifies a signal.
<i>Action</i>	Specifies a procedure.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **kill** or **killpg** subroutine, **signal** (“sigaction, sigvec, or signal Subroutine” on page 135) subroutine.

statacl or fstatacl Subroutine

Purpose

Retrieves the access control information for a file.

Library

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

Syntax

```
#include <sys/acl.h>
#include <sys/stat.h>
```

```
int statacl (Path, Command, ACL, ACLSize)
char * Path;
int Command;
struct acl * ACL;
int ACLSize;
```

```
int fstatacl (FileDescriptor, Command, ACL, ACLSize)
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.

Parameters

<i>Path</i>	Specifies a pointer to the path name of a file.
<i>FileDescriptor</i>	Specifies the file descriptor of an open file.
<i>Command</i>	Specifies the mode of the path interpretation for <i>Path</i> , specifically whether to retrieve information about a symbolic link or mount point. Valid values for the <i>Command</i> parameter are defined in the stat.h file and include: <ul style="list-style-type: none">• STX_LINK• STX_MOUNT• STX_NORMAL

ACL

Specifies a pointer to a buffer to contain the Access Control List (ACL) of the file system object. The format of an 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.

EACCES

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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **chacl** subroutine, **stat** (“statx, stat, lstat, fstatx, fstat, fullstat, fullstat, stat64, lstat64, or fstat64 Subroutine” on page 251) subroutine.

The **acl_chg** subroutine, **acl_get** subroutine, **acl_put** subroutine, **acl_set** subroutine.

The **aclget** command, **aclput** command, **chmod** command.

List of Security and Auditing Subroutines and Subroutines Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

statfs, fstatfs, or ustat Subroutine

Purpose

Gets file system statistics.

Library

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

Syntax

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

```
int statfs ( Path, StatusBuffer)
char *Path;
struct statfs *StatusBuffer;
```



```

int fstatfs ( FileDescriptor, StatusBuffer)
int FileDescriptor;
struct statfs *StatusBuffer;
#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 **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 **stat** 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.

Parameters

<i>Path</i>	The path name of any file within the mounted file system.
<i>FileDescriptor</i>	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.
<i>StatusBuffer</i>	A pointer to a statfs buffer for the returned information from the statfs or fstatfs subroutine.
<i>Device</i>	The ID of the device. It corresponds to the <i>st_rdev</i> 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.
<i>Buffer</i>	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**, 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** 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** 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".

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **stat** (“statx, stat, lstat, fstatx, fstat, fullstat, fullstat, stat64, lstat64, or fstat64 Subroutine” on page 251) subroutine.

Files, Directories, and File Systems for Programmers in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

statvfs or fstatvfs Subroutine

Purpose

Returns information about a file system.

Library

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

Syntax

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

```
int statvfs ( Path, Buf)
const char *Path;
struct statvfs *Buf;
```

```
int fstatvfs ( Fildes, Buf)
int Fildes;
struct statvfs *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 be 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.

Parameters

<i>Path</i>	The path name identifying the file.
<i>Buf</i>	A pointer to a statvfs structure in which information is returned. The statvfs structure is described in the sys/statvfs.h header file.
<i>Fildes</i>	The file descriptor identifying the open file.

Return Values

0	Successful completion.
-1	Not successful and <i>errno</i> set to one of the following.

Error Codes

EACCES	Search permission is denied on a component of the path.
---------------	---

EBADF	The file referred to by the <i>FileDescriptor</i> parameter is not an open file descriptor.
EIO	An I/O error occurred while reading from the filesystem.
ELOOP	Too many symbolic links encountered in translating path.
ENAMETOOLONG	The length of the pathname exceeds PATH_MAX , or name component is longer than NAME_MAX .
ENOENT	The file referred to by the <i>Path</i> parameter does not exist.
ENOMEM	A memory allocation failed during information retrieval.
ENOTDIR	A component of the <i>Path</i> parameter prefix is not a directory.
EOVERFLOW	One of the values to be returned cannot be represented correctly in the structure pointed to by buf .

Related Information

The **stat** (“statx, stat, lstat, fstatx, fstat, fullstat, ffullstat, stat64, lstat64, or fstat64 Subroutine”) subroutine, **statfs** (“statfs, fstatfs, or ustat Subroutine” on page 248) subroutine.

statx, stat, lstat, fstatx, fstat, fullstat, ffullstat, stat64, lstat64, or fstat64 Subroutine

Purpose

Provides information about a file.

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 Length;
```

```
int Command;
```

```
int fstatx (FileDescriptor, Buffer, Length, Command)
```

```
int FileDescriptor;
```

```
struct stat *Buffer;
```

```
int Length;
```

```
int Command;
```

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

```

int fullstat (Path, Command, Buffer)
struct fullstat *Buffer;
char *Path;
int Command;

int ffullstat (FileDescriptor, Command, Buffer)
struct fullstat *Buffer;
int FileDescriptor;
int Command;

```

Note: The **stat64**, **lstat64**, and **fstat64** subroutines apply to AIX 4.2 and later releases.

```

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;

```

Description

Note: The **stat64**, **lstat64**, and **fstat64** subroutines apply to AIX 4.2 and later releases.

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 referenced by the *FileDescriptor* parameter. The **fstatx** subroutine obtains information about the open file 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**, **fstat**, **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**, **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**.

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.

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 *st_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.

Note: The **STX_64** flag applies to AIX 4.2 and later releases.

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**.

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 <i>Path</i> or the <i>Buffer</i> parameter points to a location outside of the allocated address space of the process.
ENOENT	The file named by the <i>Path</i> parameter does not exist.
Eoverflow	The size of the file is larger than can be represented in the stat structure pointed to by the <i>Buffer</i> 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 <i>FileDescriptor</i> parameter is not a valid file descriptor.
--------------	---

EFAULT	The <i>Buffer</i> 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 <i>Length</i> value is not between 0 and the value returned by the STATSIZE macro, inclusive.
EINVAL	The <i>Command</i> parameter contains an unacceptable value.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Files

<code>/usr/include/sys/fullstat.h</code>	Contains the fullstat structure.
<code>/usr/include/sys/mode.h</code>	Defines values on behalf of the stat.h file.

Related Information

The **chmod** subroutine, **chown** subroutine, **link** subroutine, **mknod** subroutine, **mount** (“vmount or mount Subroutine” on page 380) subroutine, **openx**, **open**, or **creat** subroutine, **pipe** subroutine, **symlink** (“symlink Subroutine” on page 278) subroutine, **vtimes** subroutine.

Files, Directories, and File Systems for Programmers in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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 (“strcat, strncat, strxfrm, strcpy, strncpy, or strdup Subroutine” on page 255) (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 **strncat** 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

<i>Number</i>	Specifies the number of bytes in a string to be copied or transformed.
<i>String1</i>	Points to a string to which the specified data is copied or appended.
<i>String2</i>	Points to a string which contains the data to be copied, appended, or transformed.

Error Codes

The **strcat**, **strncat**, **strxfrm**, **strcpy**, **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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **memcpy**, **memchr**, **memcmp**, **memcpy**, or **memmove** subroutine, **setlocale** (“setlocale Subroutine” on page 107) subroutine, **strcmp**, **strncmp**, **strcasecmp**, **strncasecmp**, or **strcoll** (“strcmp, strncmp, strcasecmp, strncasecmp, or strcoll Subroutine”) subroutine, **strlen**, **strchr**, **strchr**, **strpbrk**, **strspn**, **strcspn**, **strstr**, or **strtok** (“strlen, strchr, strchr, strpbrk, strspn, strcspn, strstr, or strtok Subroutine” on page 265) subroutine, **swab** (“swab Subroutine” on page 274) subroutine.

National Language Support Overview for Programming, Understanding Multibyte and Wide Character String Collation Subroutines, Understanding Multibyte and Wide Character String Comparison Subroutines, Subroutines Overview, List of String Manipulation Services in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

strcmp, strncmp, strcasecmp, strncasecmp, or strcoll Subroutine

Purpose

Compares strings in memory.

Library

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

Syntax

```
#include <string.h>
```

```
int strcmp ( String1, String2)  
const char *String1, *String2;
```

```
int strcmp (String1, String2, Number)
const char *String1, *String2;
size_t Number;
```

```
int strcoll (String1, String2)
const char *String1, *String2;
```

```
#include <strings.h>
```

```
int strcasecmp (String1, String2)
const char *String1, *String2;
```

```
int strncasecmp (String1, String2, Number)
const char *String1, *String2;
size_t Number;
```

Description

The **strcmp**, **strncmp**, **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**, **strcasemp**, **strncasemp**, 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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **memcpy**, **memchr**, **memcmp**, **memcpy**, or **memmove** subroutine, **setlocale** (“setlocale Subroutine” on page 107) subroutine, **strcat**, **strncat**, **strxfrm**, **strcpy**, **strncpy**, or **strdup** (“strcat, strncat, strxfrm, strcpy, strncpy, or strdup Subroutine” on page 255) subroutine, **strlen**, **strchr**, **strrchr**, **strpbrk**, **strspn**, **strcspn**, **strstr**, or **strtok** (“strlen, strchr, strrchr, strpbrk, strspn, strcspn, strstr, or strtok Subroutine” on page 265) subroutine, **swab** (“swab Subroutine” on page 274) subroutine.

List of String Manipulation Subroutines, National Language Support, Multibyte and Wide Character String Collation Subroutines, Multibyte and Wide Character String Comparison Subroutines, Subroutines, Example Programs, and Libraries in *AIX 5L Version 5.1 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **perror** subroutine.

The **clearerr** macro, **feof** macro, **ferror** macro, **fileno** macro.

Subroutines Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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

<i>S</i>	Contains the output of the strfmon subroutine.
<i>MaxSize</i>	Specifies the maximum number of bytes (including the null terminating byte) that may be placed in the <i>S</i> parameter.
<i>Format</i>	Contains characters and conversion specifications.

Flags

One or more of the following flags can be specified to control the conversion:

<i>=f</i>	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.
<i>^</i>	Does not use grouping characters when formatting the currency amount. The default is to insert grouping characters if defined for the current locale.
<i>+ or (</i>	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.
<i>-</i>	Left-justifies all fields (pads to the right). The default is right-justification.
<i>!</i>	Suppresses the currency symbol from the output conversion.

Field Width

<i>w</i>	The decimal-digit string <i>w</i> specifies the minimum field width in which the result of the conversion is right-justified. If <i>-w</i> is specified, the result is left-justified. The default is a value of 0.
----------	---

Left Precision

<i>#n</i>	A # (pound sign) followed by a decimal-digit string, <i>n</i> , 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 strfmon 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 <i>n</i> variable. If more than <i>n</i> 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 <i>=f</i> 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:

```
$0000001,234.56
```

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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **scanf** ("scanf, fscanf, sscanf, or wscanf Subroutine" on page 82) subroutine, **strftime** ("strftime Subroutine") subroutine, **strptime** ("strptime Subroutine" on page 271) subroutine, **wcsftime** ("wcsftime Subroutine" on page 391) subroutine.

National Language Support Overview for Programming, Understanding Time and Monetary Formatting Subroutines, Subroutines Overview in *AIX 5L Version 5.1 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).
%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 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **localtime** subroutine, **gmtime** subroutine, **mbstowcs** subroutine, **printf** subroutine, **strfmon** (“strfmon Subroutine” on page 260) subroutine, **strptime** (“strptime Subroutine” on page 271) subroutine, **wcsftime** (“wcsftime Subroutine” on page 391) subroutine.

The **date** command.

LC_TIME Category for the Locale Definition Source File Format in *AIX 5L Version 5.1 Files Reference*.

List of Time Data Manipulation Services in *AIX 5L Version 5.1 System Management Concepts: Operating System and Devices*.

National Language Support Overview for Programming, Subroutines Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

strlen, strchr, strrchr, strpbrk, strspn, strcspn, strstr, or strtok Subroutine

Purpose

Determines the size, location, and existence of strings in memory.

Library

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

Syntax

#include <string.h>

```
size_t strlen (String)
const char * String;
```

```
char * strchr (String, Character)
const char * String;
int Character;
```

```
char * strrchr (String, Character)
const char * String;
int Character;
```

```
char * 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;
```

```
char * strstr (String1, String2)
const char * String1, * String2;
```

```
char * strtok (String1, String2)
char * String1;
const char * String2;
```

```
char *index (String, Character)
const char * String;
int Character;
```

```
char *rindex (String, Character)
const char * String;
int Character;
```

Description

Attention: Do not use the **strtok** subroutine in a multithreaded environment.

The **strlen**, **strchr**, **strrchr**, **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.

Parameters

<i>Character</i>	Specifies a character for which to return a pointer.
<i>String</i>	Points to a string from which data is returned.
<i>String1</i>	Points to a string from which an operation returns results.
<i>String2</i>	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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

The **index** and **rindex** 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.

Related Information

The **memccpy**, **memchr**, **memcmp**, **memcpy**, or **memmove** subroutine, **setlocale** (“setlocale Subroutine” on page 107) subroutine, **strcat**, **strncat**, **strxfrm**, **strcpy**, **strncpy**, or **strdup** (“strcat, strncat, strxfrm, strcpy, strncpy, or strdup Subroutine” on page 255) subroutine, **strcmp**, **strncmp**, **strcasestr**, **strncasestr**, or **strcoll** (“strcmp, strncmp, strcasestr, strncasestr, or strcoll Subroutine” on page 257) subroutine, **swab** (“swab Subroutine” on page 274) subroutine.

List of String Manipulation Services in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

National Language Support Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Subroutines Overview in *AIX 5L Version 5.1 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

```
include <string.h>
```

```
int strncollen ( String, Number)
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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **setlocale** (“setlocale Subroutine” on page 107) subroutine, **strcat**, **strncat**, **strxfrm**, **strcpy**, **strncpy**, or **strdup** (“strcat, strncat, strxfrm, strcpy, strncpy, or strdup Subroutine” on page 255) subroutine, **strcmp**, **strncmp**, **strcasestr**, **strncasestr**, or **strcoll** (“strcmp, strncmp, strcasestr, strncasestr, or strcoll Subroutine” on page 257) subroutine, **strlen**, **strchr**, **strchr**, **strpbrk**, **strspn**, **strcspn**, **strstr**, or **strtok** (“strlen, strchr, strchr, strpbrk, strspn, strcspn, strstr, or strtok Subroutine” on page 265) subroutine.

National Language Support Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

strtol, strtoul, strtoll, strtoull, atol, 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

```
#include <stdlib.h>
```

```
long strtol (String, EndPointer, Base)
```

```
const char *String;
```

```
char **EndPointer;
```

```
int Base;
```

```
unsigned long strtoul (String, EndPointer, Base)
```

```
const char *String;
```

```
char **EndPointer;
```

```
int Base;
```

```
extern long long int strtoll (String, EndPointer, Base)
```

```
char *String, **EndPointer;
```

```
int Base;
```

```
extern long long int strtoull (String, EndPointer, Base)
```

```
char *String, **EndPointer;
```

```
int Base;
```

```
long atol (String)
```

```
const char *String;
```

```
int atoi (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 **strtoll** and **strtoull** subroutines provide the same functions but return long long integers.

The **atol** subroutine is equivalent to the **strtol** subroutine where the value of the *EndPoint* parameter is a null pointer and the *Base* parameter is a value of 10.

The **atoi** subroutine is equivalent to the **strtol** subroutine where the value of the *EndPoint* parameter is a null pointer and the *Base* parameter is a value of 10.

If the value of the *EndPoint* parameter is not null, then a pointer to the character that ended the scan is stored in *EndPoint*. If an integer cannot be formed, the value of the *EndPoint* 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

<i>String</i>	Points to the character string to be converted.
<i>EndPoint</i>	Points to a character string that contains the first character not converted.
<i>Base</i>	Specifies the base to use for the conversion.

Return Values

Upon successful completion, the **strtol**, **strtoul**, **strtoll**, 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**.

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 <i>Base</i> parameter is not valid.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **atof**, **atoff**, **strtod**, or **strtof** subroutine, **scanf**, **fscanf**, **sscanf**, or **wsscanf** (“scanf, fscanf, sscanf, or wsscanf Subroutine” on page 82) subroutine, **setlocale** (“setlocale Subroutine” on page 107) subroutine, **wstrtod** or **watof** (“wstrtod or watof Subroutine” on page 457) subroutine, **wstrtol**, **watol**, or **watoi** (“wstrtol, watol, or watoi Subroutine” on page 458) subroutine.

Subroutines Overview in *AIX 5L Version 5.1 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

```
#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

<i>Buf</i>	Contains the character string to be converted by the strptime subroutine.
<i>Format</i>	Contains format specifiers for the strptime subroutine. The <i>Format</i> parameter contains 0 or more specifiers. Each specifier is composed of one of the following elements: <ul style="list-style-type: none">• 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 sign]/non-alphanumeric ordinary character.

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).
- %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).
- %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 **%l:%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.

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) 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.
- %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 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 **strptime** subroutine. If a conversion fails, the contents of the *Tm* structure are undefined.

Return Values

If successful, the **strptime** subroutine returns a pointer to the character following the last character parsed. Otherwise, a null pointer is returned.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **scanf** ("scanf, fscanf, sscanf, or wsscanf Subroutine" on page 82) subroutine, "strfmon Subroutine" on page 260, **strptime** ("strptime Subroutine" on page 262) subroutine, **time** subroutine, **wcsftime** ("wcsftime Subroutine" on page 391) subroutine.

LC_TIME Category in the Locale Definition Source File Format in *AIX 5L Version 5.1 Files Reference*.

National Language Support Overview for Programming, Understanding Time and Monetary Formatting Subroutines, Subroutines Overview in *AIX 5L Version 5.1 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>
```

```
stty ( FileDescriptor, Buffer)  
int FileDescriptor;  
struct sgttyb *Buffer;  
gtty (FileDescriptor, Buffer)  
int FileDescriptor;  
struct sgttyb *Buffer;
```

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

<i>FileDescriptor</i>	Specifies an open file descriptor.
<i>Buffer</i>	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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **ioctl** subroutine.

The Input and Output Handling Programmer's Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

swab Subroutine

Purpose

Copies bytes.

Library

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

Syntax

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

```
void swab ( From, To, NumberOfBytes)  
const void *From;  
void *To;  
ssize_t NumberOfBytes;
```

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

<i>From</i>	Points to the location of data to be copied.
<i>To</i>	Points to the array to which the data is to be copied.
<i>NumberOfBytes</i>	Specifies the number of even and nonnegative bytes to be copied.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **memccpy**, **memchr**, **memcmp**, **memmove**, or **memset** subroutine, **string** (“strlen, strchr, strrchr, strpbrk, strspn, strcspn, strstr, or strtok Subroutine” on page 265) subroutine.

List of Interfaces for Input and Output Handling in *AIX 5L Version 5.1 System Management Guide: Operating System and Devices*.

Input and Output Handling Programmer’s Overview in *AIX 5L Version 5.1 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.

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 <i>PathName</i> 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 <i>PathName</i> 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.

The Subroutines Overview in *AIX 5L Version 5.1 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

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

```
int swapon ( PathName )  
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 <i>PathName</i> 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 <i>PathName</i> 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **swapoff** subroutine, **swapqry** subroutine.

The **swapoff** command, **swapon** command.

The Subroutines Overview in *AIX 5L Version 5.1 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 <i>PathName</i> file does not exist.
ENOTBLK	Block device required.
ENOTDIR	A component of the <i>PathName</i> prefix is not a directory.
ENXIO	No such device address.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **swapoff** subroutine, **swapon** subroutine.

The **swapoff** command, **swapon** command.

Paging Space Overview in *AIX 5L Version 5.1 System Management Concepts: Operating System and Devices*.

Subroutines Overview and Understanding Paging Space Programming Requirements in *AIX 5L Version 5.1 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 15)

Applies only to symbolic links.

rename (“rename Subroutine” on page 34)

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 38)

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 369)

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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **chown**, **fchown**, **chownx**, or **fchown** subroutine, **link** subroutine, **mkdir** subroutine, **mknod** subroutine, **openx**, **open**, or **create** subroutine, **readlink** ("readlink Subroutine" on page 15) subroutine, **rename** ("rename Subroutine" on page 34) subroutine, **rmdir** ("rmdir Subroutine" on page 38) subroutine, **statx** ("statx, stat, lstat, fstatx, fstat, fullstat, fullstat, stat64, lstat64, or fstat64 Subroutine" on page 251) subroutine, **unlink** ("unlink Subroutine" on page 369) subroutine.

The **ln** command.

The **limits.h** file.

Files, Directories, and File Systems for Programmers in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

sync Subroutine

Purpose

Updates all file systems.

Library

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

Syntax

```
#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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **fsync** subroutine.

The **df** command, **sync** command.

Files, Directories, and File Systems for Programmers in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

_sync_cache_range Subroutine

Purpose

Synchronizes the I cache with the D cache.

Library

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

Syntax

```
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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The `clf` (Cache Line Flush) Instruction in *AIX 5L for POWER-based Systems 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)  
int 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.

_SC_ASYNCHRONOUS_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 **ibase** 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.

<code>_SC_CLK_TCK</code>	Indicates the clock-tick increment as defined by the CLK_TCK in the time.h file.
<code>_SC_COLL_WEIGHTS_MAX</code>	Specifies the maximum number of weights that can be assigned to an entry of the LC_COLLATE keyword in the locale definition file.
<code>_SC_DELAYTIMER_MAX</code>	Maximum number of Timer expiration overruns.
<code>_SC_EXPR_NEST_MAX</code>	Specifies the maximum number of expressions that can be nested within parentheses by the expr command. If this symbol is defined, job control is supported.
<code>_SC_JOB_CONTROL</code>	Specifies the maximum number of iovec structures one process has available for use with the readv and writev subroutines.
<code>_SC_IOV_MAX</code>	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.
<code>_SC_LINE_MAX</code>	Maximum length of a login name.
<code>_SC_LOGIN_NAME_MAX</code>	Maximum number of open message queue descriptors.
<code>_SC_MQ_OPEN_MAX</code>	Maximum number of message priorities.
<code>_SC_MQ_PRIO_MAX</code>	Implementation supports the Process Memory Locking option.
<code>_SC_MEMLOCK</code>	Implementation supports the Range Memory Locking option.
<code>_SC_MEMLOCK_RANGE</code>	Implementation supports the Memory Protection option.
<code>_SC_MEMORY_PROTECTION</code>	Implementation supports the Message Passing option.
<code>_SC_MESSAGE_PASSING</code>	Specifies the maximum number of simultaneous supplementary group IDs per process.
<code>_SC_NGROUPS_MAX</code>	Specifies the maximum number of files that one process can have open at any one time.
<code>_SC_OPEN_MAX</code>	Specifies the maximum number of significant characters in a password (not including the terminating null character).
<code>_SC_PASS_MAX</code>	Maximum number of significant bytes in a password.
<code>_SC_PASS_MAX</code>	Equivalent to <code>_SC_PAGE_SIZE</code> .
<code>_SC_PAGESIZE</code>	Size in bytes of a page.
<code>_SC_PAGE_SIZE</code>	Implementation supports the Prioritized Input and Output option.
<code>_SC_PRIORITIZED_IO</code>	Implementation supports the Process Scheduling option.
<code>_SC_PRIORITY_SCHEDULING</code>	Specifies the maximum number of repeated occurrences of a regular expression permitted when using the $\{ m, n \}$ interval notation.
<code>_SC_RE_DUP_MAX</code>	Maximum number of Realtime Signals reserved for applications use.
<code>_SC_RTSIG_MAX</code>	Implementation supports the Realtime Signals Extension option.
<code>_SC_REALTIME_SIGNALS</code>	If this symbol is defined, each process has a saved set-user ID and set-group ID.
<code>_SC_SAVED_IDS</code>	Maximum number of Semaphores per process.
<code>_SC_SEM_NSEMS_MAX</code>	Maximum value a Semaphore may have.
<code>_SC_SEM_VALUE_MAX</code>	Implementation supports the Semaphores option.
<code>_SC_SEMAPHORES</code>	Implementation supports the Shared Memory Objects option.
<code>_SC_SHARED_MEMORY_OBJECTS</code>	Maximum number of signals a process may send and have pending at any time.
<code>_SC_SIGQUEUE_MAX</code>	Specifies the maximum number of streams that one process can have open simultaneously.
<code>_SC_STREAM_MAX</code>	Implementation supports the Synchronised Input and Output option.
<code>_SC_SYNCHRONIZED_IO</code>	

<code>_SC_TIMER_MAX</code>	Maximum number of per-process Timers.
<code>_SC_TIMERS</code>	Implementation supports the Timers option.
<code>_SC_TZNAME_MAX</code>	Specifies the maximum number of bytes supported for the name of a time zone (not of the TZ value).
<code>_SC_VERSION</code>	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.
<code>_SC_XBS5_ILP32_OFF32</code>	Implementation provides a C-language compilation environment with 32-bit int, long, pointer and <code>off_t</code> types.
<code>_SC_XBS5_ILP32_OFFBIG</code>	Implementation provides a C-language compilation environment with 32-bit int, long and pointer types and an <code>off_t</code> type using at least 64 bits.
<code>_SC_XBS5_LP64_OFF64</code>	Implementation provides a C-language compilation environment with 32-bit int and 64-bit long, pointer and <code>off_t</code> types.
<code>_SC_XBS5_LPBIG_OFFBIG</code>	Implementation provides a C-language compilation environment with an int type using at least 32 bits and long, pointer and <code>off_t</code> types using at least 64 bits.
<code>_SC_XOPEN_CRYPT</code>	Indicates that the system supports the X/Open Encryption Feature Group.
<code>_SC_XOPEN_LEGACY</code>	The implementation supports the Legacy Feature Group.
<code>_SC_XOPEN_REALTIME</code>	The implementation supports the X/Open Realtime Feature Group.
<code>_SC_XOPEN_REALTIME_THREADS</code>	The implementation supports the X/Open Realtime Threads Feature Group.
<code>_SC_XOPEN_ENH_I18N</code>	Indicates that the system supports the X/Open Enhanced Internationalization Feature Group.
<code>_SC_XOPEN_SHM</code>	Indicates that the system supports the X/Open Shared Memory Feature Group.
<code>_SC_XOPEN_VERSION</code>	Indicates that the version or revision number of the X/Open standard is implemented.
<code>_SC_XOPEN_XCU_VERSION</code>	Specifies the value describing the current version of the XCU specification.
<code>_SC_ATEXIT_MAX</code>	Specifies the maximum number of register functions for the <code>atexit</code> subroutine.
<code>_SC_PAGE_SIZE</code>	Specifies page-size granularity of memory.
<code>_SC_AES_OS_VERSION</code>	Indicates OSF AES version.
<code>_SC_2_VERSION</code>	Specifies the value describing the current version of POSIX.2.
<code>_SC_2_C_BIND</code>	Indicates that the system supports the C Language binding option.
<code>_SC_2_C_CHAR_TERM</code>	Indicates that the system supports at least one terminal type.
<code>_SC_2_C_DEV</code>	Indicates that the system supports the C Language Development Utilities Option.
<code>_SC_2_C_VERSION</code>	Specifies the value describing the current version of POSIX.2 with the C Language binding.
<code>_SC_2_FORT_DEV</code>	Indicates that the system supports the FORTRAN Development Utilities Option.
<code>_SC_2_FORT_RUN</code>	Indicates that the system supports the FORTRAN Development Utilities Option.
<code>_SC_2_LOCALEDEF</code>	Indicates that the system supports the creation of locales.
<code>_SC_2_SW_DEV</code>	Indicates that the system supports the Software Development Utilities Option.
<code>_SC_2_UPE</code>	Indicates that the system supports the User Portability Utilities Option.

<code>_SC_NPROCESSORS_CONF</code>	Number of processors configured.
<code>_SC_NPROCESSORS_ONLN</code>	Number of processors online.
<code>_SC_THREAD_DATAKEYS_MAX</code>	Maximum number of data keys that can be defined in a process.
<code>_SC_THREAD_DESTRUCTOR_ITERATIONS</code>	Maximum number attempts made to destroy a thread's thread-specific data.
<code>_SC_THREAD_KEYS_MAX</code>	Maximum number of data keys per process.
<code>_SC_THREAD_STACK_MIN</code>	Minimum value for the threads stack size.
<code>_SC_THREAD_THREADS_MAX</code>	Maximum number of threads within a process.
<code>_SC_REENTRANT_FUNCTIONS</code>	System supports reentrant functions (reentrant functions must be used in multi-threaded applications).
<code>_SC_THREADS</code>	System supports POSIX threads.
<code>_SC_THREAD_ATTR_STACKADDR</code>	System supports the stack address option for POSIX threads (stackaddr attribute of threads).
<code>_SC_THREAD_ATTR_STACKSIZE</code>	System supports the stack size option for POSIX threads (stacksize attribute of threads).
<code>_SC_THREAD_PRIORITY_SCHEDULING</code>	System supports the priority scheduling for POSIX threads.
<code>_SC_THREAD_PRIO_INHERIT</code>	System supports the priority inheritance protocol for POSIX threads (priority inversion protocol for mutexes).
<code>_SC_THREAD_PRIO_PROTECT</code>	System supports the priority ceiling protocol for POSIX threads (priority inversion protocol for mutexes).
<code>_SC_THREAD_PROCESS_SHARED</code>	System supports the process sharing option for POSIX threads (pshared attribute of mutexes and conditions).
<code>_SC_TTY_NAME_MAX</code>	Maximum length of a terminal device name.
Note: The <code>_SYNCHRONIZED_IO</code> , <code>_SC_FSYNC</code> , and <code>SC_MAPPED_FILES</code> commands apply to operating system version 4.3 and later releases.	
<code>_SC_SYNCHRONIZED_IO</code>	Implementation supports the Synchronized Input and Output option.
<code>_SC_FSYNC</code>	Implementation supports the File Synchronization option.
<code>_SC_MAPPED_FILES</code>	Implementation supports the Memory Mapped Files option.

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**.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

File

`/usr/include/limits.h`

Contains system-defined limits.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The `confstr` subroutine, `pathconf` subroutine.

The `bc` command, `expr` command.

Subroutines Overview in *AIX 5L Version 5.1 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

```
#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 291)	Loads a kernel extension object file into kernel memory.
SYS_SINGLELOAD ("SYS_SINGLELOAD sysconfig Operation" on page 301)	Loads a kernel extension object file only if it is not already loaded.

SYS_QUERYLOAD ("SYS_QUERYLOAD sysconfig Operation" on page 298)	Determines if a specified kernel object file is loaded.
SYS_KULOAD ("SYS_KULOAD sysconfig Operation" on page 296)	Unloads a previously loaded kernel object file.
SYS_QDVSW ("SYS_QDVSW sysconfig Operation" on page 297)	Checks the status of a device switch entry in the device switch table.
SYS_CFGDD ("SYS_CFGDD sysconfig Operation" on page 288)	Calls the specified device driver configuration routine (module entry point).
SYS_CFGKMOD ("SYS_CFGKMOD sysconfig Operation" on page 289)	Calls the specified module at its module entry point for configuration purposes.
SYS_GETPARMS ("SYS_GETPARMS sysconfig Operation" on page 290)	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 299)	Sets run-time system parameters from a caller-provided structure.

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 292 explains the symbol binding support provided when loading kernel object files.

Parameters

<i>Cmd</i>	Specifies the function that the sysconfig subroutine is to perform.
<i>Parmp</i>	Specifies a user-provided structure.
<i>Parmlen</i>	Specifies the length of the user-provided structure indicated by the <i>Parmp</i> 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 to **EFAULT**.

Related Information

The **ddconfig** device driver entry point.

Device Configuration Subsystem, Kernel Environment, Understanding Kernel Extension Binding in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*.

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 *parm* 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.dds_ptr* and *cfg_dd.dds_len* 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 <i>parm</i> and <i>parmlen</i> 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** (“sysconfig Subroutine” on page 286) subroutine.

The **ddconfig** device driver entry point.

The **uio** structure.

Device Configuration Subsystem Programming Introduction, Device Dependent Structure (DDS) Overview, Device Driver Kernel Extension Overview, Programming in the Kernel Environment Overview, Understanding Kernel Extension Binding, Understanding the Device Switch Table in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*.

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** (“SYS_CFGDD sysconfig Operation” on page 288) 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** (“SYS_KLOAD sysconfig Operation” on page 291) or **SYS_SINGLELOAD** (“SYS_SINGLELOAD sysconfig Operation” on page 301) 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 *mdiptr* 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 *mdiptr* and *mdilen* 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:

```
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 <i>parmp</i> and <i>parmlen</i> 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 286) subroutine.

The **SYS_CFGDD** (“SYS_CFGDD sysconfig Operation” on page 288) sysconfig operation, **SYS_KLOAD** (“SYS_KLOAD sysconfig Operation” on page 291) sysconfig operation, **SYS_SINGLELOAD** (“SYS_SINGLELOAD sysconfig Operation” on page 301) sysconfig operation.

The **uio** structure.

Device Configuration Subsystem Programming Introduction in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*

Programming in the Kernel Environment Overview in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*

Understanding Kernel Extension Binding in *AIX 5L Version 5.1 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 299) 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 286) subroutine and **sys_parm** (“sys_parm Subroutine” on page 308) subroutine.

The **SYS_SETPARMS** (“SYS_SETPARMS sysconfig Operation” on page 299) sysconfig operation.

Programming in the Kernel Environment Overview in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*.

SYS_KLOAD sysconfig Operation

The **SYS_KLOAD** sysconfig operation includes information for **SYS_KLOAD** on a POWER-based platform and an **SYS_KLOAD** on an Itanium-based platform.

SYS_KLOAD sysconfig operation on POWER-based Platform

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** (“SYS_CFGKMOD sysconfig Operation” on page 289) operation)

- Invocation of a device driver's **ddconfig** subroutine (using the **SYS_CFGDD** ("SYS_CFGDD sysconfig Operation" on page 288) operation)
- Unloading the kernel module (using the **SYS_KULOAD** ("SYS_KULOAD sysconfig Operation" on page 296) operation).

The *parm* 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** ("SYS_SINGLELOAD sysconfig Operation" on page 301) 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: <ul style="list-style-type: none">• 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 <i>parmp</i> and <i>parmlen</i> 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** (“sysconfig Subroutine” on page 286) subroutine.

The **SYS_SINGLELOAD** (“SYS_SINGLELOAD sysconfig Operation” on page 301) sysconfig operation, **SYS_KULOAD** (“SYS_KULOAD sysconfig Operation” on page 296) sysconfig operation, **SYS_CFGDD** (“SYS_CFGDD sysconfig Operation” on page 288) sysconfig operation, **SYS_CFGKMOD** (“SYS_CFGKMOD sysconfig Operation” on page 289) sysconfig operation.

The **ddconfig** device driver entry point.

Device Configuration Subsystem Programming Introduction in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*.

Programming in the Kernel Environment Overview in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*.

Understanding Kernel Extension Binding in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*.

SYS_KLOAD sysconfig operation on Itanium-based Platform

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** ("SYS_CFGKMOD sysconfig Operation" on page 289) operation)
- Invocation of a device driver's **ddconfig** subroutine (using the **SYS_CFGDD** ("SYS_CFGDD sysconfig Operation" on page 288) operation)
- Unloading the kernel module (using the **SYS_KULOAD** ("SYS_KULOAD sysconfig Operation" on page 296) 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** ("SYS_SINGLELOAD sysconfig Operation" on page 301) 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 *%soname /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: Dependent modules must have a fully qualified base filename.

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.

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: <ul style="list-style-type: none">• 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 <i>parmp</i> and <i>parmlen</i> 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 ELF object file indication in its header. The SYS_KLOAD operation only supports loading of ELF object files. This error is also returned if the loader is unable to resolve an imported symbol.
EINVAL	The program file has a valid ELF 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** (“sysconfig Subroutine” on page 286) subroutine.

The **SYS_SINGLELOAD** (“SYS_SINGLELOAD sysconfig Operation” on page 301) sysconfig operation, **SYS_KULOAD** (“SYS_KULOAD sysconfig Operation” on page 296) sysconfig operation, **SYS_CFGDD** (“SYS_CFGDD sysconfig Operation” on page 288) sysconfig operation, **SYS_CFGKMOD** (“SYS_CFGKMOD sysconfig Operation” on page 289) sysconfig operation.

The **ddconfig** device driver entry point.

Device Configuration Subsystem Programming Introduction in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*.

Programming in the Kernel Environment Overview in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*.

Understanding Kernel Extension Binding in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*.

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 291) 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 301) 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 288) 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 294 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 <i>parmp</i> and <i>parmlen</i> 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 **SYS_CFGDD** (“SYS_CFGDD sysconfig Operation” on page 288) sysconfig operation, **SYS_KLOAD** (“SYS_KLOAD sysconfig Operation” on page 291) sysconfig operation, **SYS_SINGLELOAD** (“SYS_SINGLELOAD sysconfig Operation” on page 301) sysconfig operation.

The **sysconfig** (“sysconfig Subroutine” on page 286) subroutine.

Device Configuration Subsystem Programming Introduction in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*.

Programming in the Kernel Environment Overview in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*.

Understanding Kernel Extension Binding in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*.

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 *parm* 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 87) 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 37) 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 *parm* 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 286) 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, **ddrevoke** device driver entry point.

The **console** special file.

Device Configuration Subsystem Programming Introduction in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*.

Programming in the Kernel Environment Overview in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*.

Understanding Kernel Extension Binding in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*.

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 291) 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 294 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:

- 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.
- EFAULT** The *path* parameter points to a location outside of the allocated address space of the process.
- EIO** An I/O error occurred during the operation.

Related Information

The **sysconfig** (“sysconfig Subroutine” on page 286) subroutine.

The **SYS_SINGLELOAD** (“SYS_SINGLELOAD sysconfig Operation” on page 301) sysconfig operation, **SYS_KLOAD** (“SYS_KLOAD sysconfig Operation” on page 291) sysconfig operation.

Programming in the Kernel Environment Overview in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*.

Understanding Kernel Extension Binding Overview in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*.

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 290) operation and the data written by the **SYS_SETPARMS** operation.

The `parm_p` 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 `parm_len` 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 `parm_len` 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: <ul style="list-style-type: none">• The <code>var_vers</code> version number of the provided structure does not match the version number of the current <code>var</code> structure.• The structure provided by the caller does not contain enough data to specify the header fields within the <code>var</code> structure.• One of the specified variable values is invalid or not allowed. On the return from the subroutine, the <code>var_vers</code> field in the caller-provided buffer contains the byte offset of the first variable in the structure that was detected in error.
EAGAIN	The <code>var_gen</code> 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 <code>parm_p</code> and <code>parm_len</code> 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 286) subroutine and **sys_parm** (“sys_parm Subroutine” on page 308) subroutine.

The **SYS_GETPARMS** (“SYS_GETPARMS sysconfig Operation” on page 290) 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** (“SYS_KLOAD sysconfig Operation” on page 291) 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” on page 294 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** (“sysconfig Subroutine” on page 286) subroutine.

The **SYS_KLOAD** (“SYS_KLOAD sysconfig Operation” on page 291) sysconfig operation.

Programming in the Kernel Environment Overview, and Understanding Kernel Extension Binding in *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*.

syslog, openlog, closelog, or setlogmask Subroutine

Purpose

Controls the system log.

Library

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

Syntax

```
#include <syslog.h>
```

```
void openlog ( ID, LogOption, Facility)
const char *ID;
int LogOption, Facility;
```

```
void syslog ( Priority, Value,... )
int Priority;
const char *Value;
```

```
void closelog ( )
```

```
int setlogmask( MaskPriority)
int MaskPriority;
```

```
void bsdlog (Priority, Value,...)
int Priority;
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** (“**syslog_r**, **openlog_r**, **closelog_r**, or **setlogmask_r** Subroutine” on page 305), **openlog_r**, **closelog_r**, or **setlogmask_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.

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_RFS

Logs the remote file systems (Andrew File System and RVD).

LOG_UUCP

Logs the UUCP subsystem.

LOG_USER

Logs messages generated by user processes. This is the default facility when none is specified.

<i>Priority</i>	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 Indicated 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.
<i>MaskPriority</i>	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.
<i>Value</i>	Specifies the values given in the <i>Value</i> parameters and follows the the same syntax as the printf subroutine <i>Format</i> parameter.

Examples

- To log an error message concerning a possible security breach, such as the following, enter:

```
syslog (LOG_ALERT, "who:internal error 23");
```
- 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);
```
- To log an error message from the system, enter:

```
syslog (LOG_INFO | LOG_LOCAL2, "foobar error: %m");
```

Implementation Specifics

These subroutines are part of the operating system.

Related Information

The **profil** subroutine.

The **prof** command.

The **syslogd** daemon.

_end, **_etext**, or **edata** identifiers.

Subroutines Overview in *AIX 5L Version 5.1 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.

Parameters

<i>Priority</i>	<p>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:</p> <ul style="list-style-type: none">• 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.
<i>SysLogData</i>	<p>Specifies a structure that contains the following information:</p> <ul style="list-style-type: none">• 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.
<i>Format</i>	<p>Specifies the format, given in the same format as for the printf subroutine.</p>
<i>ID</i>	<p>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.</p>
<i>LogOption</i>	<p>Specifies a bit field that indicates logging options. The values of <i>LogOption</i> are:</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>LOG_ODELAY Delays opening until the syslog subroutine is called.</p> <p>LOG_PID Logs the process ID with each message. This option is useful for identifying daemons.</p>

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_RFS

Logs the remote file systems (Andrew File System and RVD).

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.

Examples

1. To log an error message concerning a possible security breach, enter:

```
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:

```
openlog_r ("ftpd", LOG_PID, LOG_DAEMON, syslog_data_struct);  
setlogmask_r (LOG_UPTO (LOG_ERR), syslog_data_struct);  
syslog_r (LOG_INFO, syslog_data_struct, "");
```
3. To log an error message from the system, enter:

```
syslog_r (LOG_INFO | LOG_LOCAL2, syslog_data_struct, "system error: %m");
```

Implementation Specifics

These subroutines are part of the operating system.

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

Related Information

The **prof** command.

The **syslogd** daemon.

The **printf**, **fprintf**, **sprintf**, **wsprintf**, **vprintf**, **vfprintf**, **vsprintf**, or **vwsprintf** subroutine.

Subroutines Overview and List of Multithread Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

sys_parm Subroutine

Purpose

Provides a service for examining or setting kernel run-time tunable parameters.

Library

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

Syntax

```
#include <sys/types.h>
#include <sys/var.h>
```

```
int sys_parm ( cmd, parmflag, parm)
int cmd;
int parmflag;
struct vario *parm;
```

Description

The **sys_parm** subroutine is used to query and/or customize run-time operating system parameters.

Note: This is a replacement service for **sysconfig** with respect to querying or changing information in the **var** structure.

The **sys_parm** subroutine:

- Works on both 32 bit and 64 bit platforms
- Requires appropriate privilege for its use.

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

parmflag

parmp

Specifies the **SYSP_GET** or **SYSP_SET** function.

Specifies the parameter upon which the function will act.

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).

```
#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).

```
#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

EINVAL

The calling process does not have the required privilege.

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.

File

sys/var.h

Contains structure definitions.

Related Information

The **SYS_GETPARMS** (“SYS_GETPARMS sysconfig Operation” on page 290) **sysconfig** Operation, and **SYS_SETPARMS** (“SYS_SETPARMS sysconfig Operation” on page 299) **sysconfig** Operation

system Subroutine

Purpose

Runs a shell command.

Library

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

Syntax

```
#include <stdlib.h>
```

```
int system ( String )  
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 I** 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **execl** subroutine, **exit** subroutine, **fork** subroutine, **pipe** subroutine, **wait** (“wait, waitpid, wait3, or wait364 Subroutine” on page 385) subroutine, **waitpid** (“wait, waitpid, wait3, or wait364 Subroutine” on page 385) subroutine.

The **sh** command.

List of Security and Auditing Subroutines, Subroutines Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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 5.1 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **chmod** or **fchmod** subroutine, **statx**, **stat**, **lstat**, **fstatx**, **fstat**, **fullstat**, or **ffullstat** (“statx, stat, lstat, fstatx, fstat, fullstat, ffullstat, stat64, lstat64, or fstat64 Subroutine” on page 251) subroutine.

The **chmod** command.

List of Security and Auditing Subroutines, Subroutines Overview in *AIX 5L Version 5.1 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)  
int 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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **tcflow** (“tcflow Subroutine”) subroutine, **tcflush** (“tcflush Subroutine” on page 315) subroutine, **tcsendbreak** (“tcsendbreak Subroutine” on page 318) subroutine.

The Input and Output Handling Programmer’s Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

tcflow Subroutine

Purpose

Performs flow control functions.

Library

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

Syntax

```
#include <termios.h>
```

```
int tcflow( FileDescriptor, 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

<i>FileDescriptor</i>	Specifies an open file descriptor.
<i>Action</i>	Specifies one of the following:
TCOOFF	Suspend output.
TCOON	Restart suspended output.
TCIOFF	Transmit a STOP character, which is intended to cause the terminal device to stop transmitting data to the system. See the description of IXOFF in the Input Modes section of the termios.h file.
TCION	Transmit a START character, which is intended to cause the terminal device to start transmitting data to the system. See the description of IXOFF in the Input Modes section of the termios.h file.

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 <i>FileDescriptor</i> parameter does not specify a valid file descriptor.
EINVAL	The <i>Action</i> 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 <i>FileDescriptor</i> parameter is not a terminal.

Example

To restart output from a terminal device, enter:

```
rc = tcflow(stdout, TCION);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **tcdrain** (“tcdrain Subroutine” on page 312) subroutine, **tcflush** (“tcflush Subroutine” on page 315) subroutine, **tcsendbreak** (“tcsendbreak Subroutine” on page 318) subroutine.

tcflush Subroutine

Purpose

Discards data from the specified queue.

Library

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

Syntax

```
#include <termios.h>
```

```
int tcflush( FileDescriptor, QueueSelector)
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
QueueSelector

Specifies an open file descriptor.

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 <i>FileDescriptor</i> parameter does not specify a valid file descriptor.
EINVAL	The <i>QueueSelector</i> 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 <i>FileDescriptor</i> parameter is not a terminal.

Example

To flush the output queue, enter:

```
rc = tcflush(2, TCOFLUSH);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **tcdrain** (“tcdrain Subroutine” on page 312) subroutine, **tcflow** (“tcflow Subroutine” on page 313) subroutine, **tcsendbreak** (“tcsendbreak Subroutine” on page 318) subroutine.

The Input and Output Handling Programmer’s Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

tcgetattr Subroutine

Purpose

Gets terminal state.

Library

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

Syntax

```
#include <termios.h>
```

```
int tcgetattr ( FileDescriptor, TermiosPointer)
int FileDescriptor;
struct termios *TermiosPointer;
```

Description

The **tcgetattr** subroutine gets the parameters associated with the object referred to by the *FileDescriptor* parameter and stores them in the **termios** structure referenced by the *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 **termios** structure returned by the **tcgetattr** subroutine reflect the actual baud rates, even if they are equal.

Note: If differing baud rates are not supported, returning a value of 0 as the input baud rate is obsolete.

Parameters

<i>FileDescriptor</i>	Specifies an open file descriptor.
<i>TermiosPointer</i>	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 **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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **tcsetattr** (“tcsetattr Subroutine” on page 319) subroutine.

The Input and Output Handling Programmer’s Overview in *AIX 5L Version 5.1 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 ( FileDescriptor )  
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 **tcsetpgrp** 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **setpgid** (“setpgid or setpgrp Subroutine” on page 117) subroutine, **setsid** (“setsid Subroutine” on page 122) subroutine, **tcsetpgrp** (“tcsetpgrp Subroutine” on page 321) subroutine.

The Input and Output Handling Programmer’s Overview in *AIX 5L Version 5.1 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( FileDescriptor, Duration )  
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.

Parameters

<i>FileDescriptor</i>	Specifies an open file descriptor.
<i>Duration</i>	Specifies the number of milliseconds that zero-valued bits are transmitted. If the value of the <i>Duration</i> parameter is 0, it causes transmission of zero-valued bits for at least 250 milliseconds and not longer than 500 milliseconds. If <i>Duration</i> is not 0, it sends zero-valued bits for <i>Duration</i> 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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Pseudo-terminals and LFT do not generate a break condition. They return without taking any action.

Related Information

The **tcdrain** (“tcdrain Subroutine” on page 312) subroutine, **tcflow** (“tcflow Subroutine” on page 313) subroutine, **tcflush** (“tcflush Subroutine” on page 315) subroutine.

The Input and Output Handling Programmer’s Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

tcsetattr Subroutine

Purpose

Sets terminal state.

Library

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

Syntax

```
#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

<i>FileDescriptor</i>	Specifies an open file descriptor.
<i>OptionalActions</i>	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 <i>FileDescriptor</i> 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 <i>FileDescriptor</i> has been transmitted. All input that has been received but not read is discarded before the change is made.
<i>TermiosPointer</i>	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.

EINTR	A signal interrupted the tcsetattr subroutine.
EINVAL	The <i>OptionalActions</i> 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 <i>FileDescriptor</i> parameter is not a terminal.

Example

To set the terminal state after the current output completes, enter:

```
rc = tcsetattr(stdout, TCSADRAIN, &my_termios);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **cfgetispeed** subroutine, **tcgetattr** (“tcgetattr Subroutine” on page 316) subroutine.

The Input and Output Handling Programmer’s Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

tcsetpgrp Subroutine

Purpose

Sets foreground process group ID.

Library

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

Syntax

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

```
int tcsetpgrp ( FileDescriptor, ProcessGroupID)
int FileDescriptor;
pid_t ProcessGroupID;
```

Description

If the process has a controlling terminal, the **tcsetpgrp** subroutine sets the foreground process group ID associated with the terminal to the value of the *ProcessGroupID* parameter. The file associated with the *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 *ProcessGroupID* parameter must match a process group ID of a process in the same session as the calling process.

Parameters

<i>FileDescriptor</i>	Specifies an open file descriptor.
<i>ProcessGroupID</i>	Specifies the process group identifier.

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

This function is unsuccessful if one of the following is true:

EBADF	The <i>FileDescriptor</i> parameter is not a valid file descriptor.
EINVAL	The <i>ProcessGroupID</i> parameter is invalid.
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.
EPERM	The <i>ProcessGroupID</i> parameter is valid, but does not match the process group ID of a process in the same session as the calling process.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **tcgetpgrp** (“tcgetpgrp Subroutine” on page 317) subroutine.

The Input and Output Handling Programmer’s Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

termdef Subroutine

Purpose

Queries terminal characteristics.

Library

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

Syntax

```
char *termdef ( FileDescriptor, Characteristic )
int FileDescriptor;
char Characteristic;
```

Description

The **termdef** 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 **termdef** 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

OR

- Your asynchronous terminal has an unusual number of lines or columns and you are running an application that uses the **termdef** 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 568), calls the **termdef** subroutine to determine 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.

Parameters

FileDescriptor
Characteristic

Specifies an open file descriptor.

Specifies the characteristic that is to be queried. The following values can be specified:

- c** Causes the **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 **winsize** structure by issuing the **TIOCGWINSZ ioctl**. If **ws_col** is not 0, the **ws_col** value is used.
 2. If the **TIOCGWINSZ ioctl** is unsuccessful or if **ws_col** is 0, the **termdef** subroutine attempts to use the value of the **COLUMNS** environment variable.
 3. If the **COLUMNS** environment variable is not set, the **termdef** subroutine returns a pointer to a null string.
- l** Causes the **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 **winsize** structure by issuing the **TIOCGWINSZ ioctl**. If **ws_row** is not 0, the **ws_row** value is used.
 2. If the **TIOCGWINSZ ioctl** is unsuccessful or if **ws_row** is 0, the **termdef** subroutine attempts to use the value of the **LINES** environment variable.
 3. If the **LINES** environment variable is not set, the **termdef** subroutine returns a pointer to a null string.

Characters other than c or l

Cause the **termdef** subroutine to query for the “terminal type” of the terminal. This is determined by performing the following actions:

1. The **termdef** subroutine attempts to use the value of the **TERM** environment variable.
2. If the **TERM** environment variable is not set, the **termdef** subroutine returns a pointer to string set to “dumb”.

Examples

1. To display the terminal type of the standard input device, enter:

```
printf("%s\n", termdef(0, 't'));
```

2. To display the current lines and columns of the standard output device, enter:

```
printf("lines\tcolumns\n%s\t%s\n", termdef(2, 'l'),
      termdef(2, 'c'));
```

Note: If the **termdef** subroutine is unable to determine a value for lines or columns, it returns pointers to null strings.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **setupterm** (“setupterm Subroutine” on page 568) subroutine.

The **stty** command.

The Input and Output Handling Programmer’s Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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

```
#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

<i>zone</i>	Specifies minutes westward from Greenwich.
<i>dst</i>	Specifies whether to return Standard time or Daylight Savings time.
<i>czone</i>	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.

Implementations Specifics

These subroutines are part of Base Operation System (BOS) Runtime.

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

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

```
int thread_post( tid)
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.
EPERM	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` (“thread_wait Subroutine” on page 329) subroutine, and `thread_post_many` (“thread_post_many Subroutine”) 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)
int nthreads;
tid_t * tidp;
tid_t * erridp;
```

Description

The `thread_post_many` subroutine posts one or more threads of the occurrence of the event. The number of threads to be posted is specified by the value of the *nthreads* parameter, while the *tidp* parameter points to an array of thread IDs of threads that need to be posted. The subroutine works just like the `thread_post` subroutine but can be used to post to multiple threads at the same time.

A maximum of 512 threads can be posted in one call to the `thread_post_many` subroutine.

An optional address to a thread ID field may be passed in the *erridp* parameter. This field is normally ignored by the kernel unless the subroutine fails because the calling process has no permissions to post to any one of the specified threads. In this case, the kernel posts all threads in the array pointed at by the *tidp* parameter up to the first failing thread and fills the *erridp* parameter with the failing thread's ID.

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:

ESRCH

None of the indicated threads are existent or they have all exited or are exiting.

EPERM

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.

EFAULT

The *tidp* parameter points to a location outside of the address space of the process.

EINVAL

A negative value or a value greater than 512 was specified in the *nthreads* parameter.

Related Information

The **thread_wait** (“thread_wait Subroutine” on page 329) subroutine, and **thread_post** (“thread_post Subroutine” on page 325) subroutine.

thread_self Subroutine

Purpose

Returns the caller’s kernel thread ID.

Library

Standard C library (**libc.a**)

Syntax

```
#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 (**libpthread.a**) and returned by the **pthread_self** subroutine.

Return Values

The **thread_self** subroutine returns the caller’s kernel thread ID.

Implementation Specifics

This subroutine is part of the Base Operating System (BOS) Runtime.

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.

Implementation Specifics

This subroutine is part of the Base Operating System (BOS) Runtime.

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( timeout)
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** (“thread_post Subroutine” on page 325) subroutine, and **thread_post_many** (“thread_post_many Subroutine” on page 326) 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **fopen**, **freopen**, **fdopen** subroutines, **mktemp** subroutine, **tmpnam** or **tempnam** (“tmpnam or tempnam Subroutine” on page 331) subroutine, **unlink** (“unlink Subroutine” on page 369) subroutine.

tmpnam or tempnam Subroutine

Purpose

Constructs the name for a temporary file.

Library

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

Syntax

```
#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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **fopen**, **freopen**, **fdopen** subroutine, **malloc**, **free**, **realloc**, **calloc**, **mallopt**, **mallinfo**, or **alloca** subroutine, **mktemp** or **mkstemp** subroutine, **openx**, **open**, **creat** subroutine, **tmpfile** (“tmpfile Subroutine” on page 330) subroutine, **unlink** (“unlink Subroutine” on page 369) subroutine.

The **environment** file.

Files, Directories, and File Systems for Programmers in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Input and Output Handling Programmer's Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

towctrans Subroutine

Purpose

Character transliteration.

Library

Standard library (**libc.a**)

Syntax

```
#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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) subroutine.

Related Information

The **towlower** (“towlower Subroutine”) subroutine, **towupper** (“towupper Subroutine” on page 334) subroutine, **wctrans** (“wctrans Subroutine” on page 413) 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)
wint_t 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **iswalnum** subroutine, **iswalph** subroutine, **iswcntrl** subroutine, **iswctype** subroutine, **iswdigit** subroutine, **iswgraph** subroutine, **iswlower** subroutine, **iswprint** subroutine, **iswpunct** subroutine, **iswspace** subroutine, **iswupper** subroutine, **iswxdigit** subroutine, **setlocale** (“setlocale Subroutine” on page 107) subroutine, **towupper** (“towupper Subroutine”) subroutine, **wctype** (“wctype or get_wctype Subroutine” on page 413) subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Wide Character Classification Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

towupper Subroutine

Purpose

Converts a lowercase wide character to an uppercase wide character.

Library

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

Syntax

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

```
wint_t towupper ( WC)  
wint_t 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

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** (“setlocale Subroutine” on page 107) subroutine, **towlower** (“towlower Subroutine” on page 333) subroutine, **wctype** (“wctype or get_wctype Subroutine” on page 413) subroutine.

Subroutines Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

t_rcvreldata Subroutine

Purpose

Receive an orderly release indication or confirmation containing user data.

Library

Syntax

```
#include <xti.h>

int t_rcvreldata(
    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 337). 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 343) 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_rcvreldata** 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.

Parameters	Before call	After call
fd	x	/
discon->	udata.maxlen	x
discon->	udata.len	/
discon->	udata.buf	?
discon->	reason	/
discon->	sequence	/

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 **t_errno** is set to indicate an error.

Error Codes

On failure, the **t_errno** subroutine is set to one of the following:

TBADF

The specified file descriptor does not refer to a transport endpoint.

TBUFOVFLW

The number of bytes allocated for incoming data (**maxlen**) is greater than 0 but not sufficient to store the data, and the disconnection information to be returned in **discon** will be discarded. The provider state, as seen by the user, will be changed as if the data was successfully retrieved.

TLOOK

An asynchronous event has occurred on this transport endpoint and requires immediate attention.

TNOREL

No orderly release indication currently exists on the specified transport endpoint.

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.

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 **t_getinfo**, **t_open**, **t_sndreldata** (“t_sndreldata Subroutine” on page 343), **t_rcvrel**, **t_sndrel** subroutines.

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

```
#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 *fd* identifies the local transport endpoint through which data will arrive, *iov* points to an array of buffer address/buffer size pairs (*iov_base*, *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 [TBADDDATA].

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 **ort_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.

Parameters	Before call	After call
fd	X	/
iov	X/	
iovcount	X	/
iov[0].iov_base	X(/)	=(X)
iov[0].iov_len	X	=
...		
iov[iovcount-1].iov_base	X(/)	=(X)
iov[iovcount-1].iov_len	X	=

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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) subroutine.

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 341) subroutine.

t_rcvvudata Subroutine

Purpose

Receive a data unit into one or more noncontiguous buffers.

Library

Standard library (**libxti.a**)

Syntax

```
#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 [TBADDDATA]. 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 userdata 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	Before call	After call
fd	X	/
unitdata->addr.maxlen	X	=

Parameters	Before call	After call
unitdata->addr.len	/	X
unitdata->addr.buf	?(/)	=(/)
unitdata->opt.maxlen	X	=
unitdata->opt.len	/	X
unitdata->opt.buf	?(/)	=(?)
unitdata->udata.maxlen	/	=
unitdata->udata.len	/	=
unitdata->udata.buf	/	=
iov[0].iov_base	X	=(X)
iov[0].iov_len	X	=
. . . .		
iov[iovcount-1].iov_base	X(/)	=(X)
iov[iovcount-1].iov_len	X	=
iovcount	X	/
flags	/	/

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.
TBADF	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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) subroutine.

Related Information

The **fcntl** subroutine, **t_alloc** subroutine, **t_open** subroutine, **t_rcvvudata** subroutine, **t_rcvuderr** subroutine, **t_sndudata** subroutine, **t_sndvudata** (“t_sndvudata Subroutine” on page 345) 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

```
#include <xti.h>
int t_sndv (int fd, const struct t_iovec *iov, unsigned int iovcount, int flags)
```

Description

Parameters	Before call	After call
fd	X	/
iov	X	/
iovcount	X	/
iov[0].iov_base	X(X)	/
iov[0].iov_len	X	/
...		
iov[iovcount-1].iov_base	X(X)	/
iov[iovcount-1].iov_len	X	=
flags	X	/

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/buffer 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 [TBADDDATA].

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 **ort_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[iovcnt-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, **t_sndv** returns a value that is less than the value of nbytes. If **t_sndv** is interrupted by a signal before it could transfer data to the communications provider, it returns -1 with **t_errno** set to [TYSERR] and **errno** set to [EINTR].

If the number of bytes of data in the iov array is zero and sending of zero octets is not supported by the underlying transport service, **t_sndv** returns -1 with **t_errno** set to [TBADDDATA].

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 info argument returned by **t_getinfo**.

The error [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, **t_sndv** returns the number of bytes accepted by the transport provider. Otherwise, -1 is returned on failure and **t_errno** is set to indicate the error.

Notes:

1. In synchronous mode, if more than INT_MAX bytes of data are passed in the iov array, only the first 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 `O_NONBLOCK` is set and the communications provider is blocked due to flow control, or that `O_NONBLOCK` is clear and the function was interrupted by a signal.

Error Codes

On failure, `t_errno` is set to one of the following:

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 **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 **info** argument the ability of an XTI implementation to detect such an error case is implementation-dependent (see *CAVEATS*, below).
- **iovcount** is greater than `T_IOV_MAX`.

TBADF The specified file descriptor does not refer to a transport endpoint.

TBADFLAG An invalid flag was specified.

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 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) subroutine.

Related Information

The **t_getinfo** subroutine, **t_open** subroutine, **t_rcvv** (“**t_rcvv** Subroutine” on page 337) subroutine, **t_rcv** subroutine, **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:

```
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.

Parameters	Before call	After call
fd	x	/
discon->	udata.maxlen	/
discon->	udata.len	x
discon->	udata.buf	?(?)
discon->	reason	?
discon->	sequence	/

Valid States

T_DATAXFER, T_INREL

Error Codes

On failure, **t_errno** is set to one of the following:

[TBADDDATA]

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.

[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 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 335), **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:

```
struct netbuf addr;  
struct netbuf opt;  
struct netbuf udata;
```

If the limit on **iovcount** is exceeded, the function fails with [TBADDDATA].

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]**.

The limit on the total number of bytes available in all buffers passed (that is:

$iov(0).iov_len + \dots + iov(iovcount-1).iov_len$)

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 [TBADDDATA] 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 [TBADDDADDR] and [TBADDOPT], 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	Before call	After call
fd	x	/
unitdata->	addr.maxlen	/
unitdata->	addr.len	x
unitdata->	addr.buf	x(x)
unitdata->	opt.maxlen	/
unitdata->	opt.len	x
unitdata->	opt.buf	?(?)
unitdata->	udata.maxlen	/
unitdata->	udata.len	/
unitdata->	udata.buf	/
iov[0].iov_base	x(x)	=(=)
left>iov[0].iov_len	x	=
...		
iov[iovcount-1].iov_base	x(x)	=(=)
iov[iovcount-1].iov_len	x	=
iovcount	x	/

Valid States

T_IDLE

Error Codes

On failure, **t_errno** is set to one of the following:

[TBADADDR] [TBADDDATA]	The specified protocol address was in an incorrect format or contained illegal information. Illegal amount of data. <ul style="list-style-type: none"> A single send was attempted specifying a TSDU greater than that specified in the <i>info</i> 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.
[TBADDOPT]	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]	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.

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_rcvvudata Subroutine” on page 339), **t_rcvuderr**, **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

Parameters	Before call	After call
name	X	/

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**.

Variable	Value of Name
T_IOV_MAX	_SC_T_IOV_MAX

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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) subroutine.

Related Information

The **t_rcvv** (“t_rcvv Subroutine” on page 337) subroutine, **t_rcvvudata** (“t_rcvvudata Subroutine” on page 339) subroutine, **t_sndv** (“t_sndv Subroutine” on page 341) subroutine, **t_sndvudata** (“t_sndvudata Subroutine” on page 345) subroutine.

trcgen or trcgent Subroutine

Purpose

Records a trace event for a generic trace channel.

Library

Runtime Services Library (**librts.a**)

Syntax

```
#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

<i>Buffer</i>	Specifies a pointer to a buffer of trace data. The maximum size of the trace data is 4096 bytes.
<i>Channel</i>	Specifies a channel number for the trace session, obtained from the trcstart subroutine.
<i>DataWord</i>	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:

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.

Length Specifies the length in bytes of the *Buffer* parameter.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **trchook** (“trchook or utrchook Subroutine”) subroutine, **trcoff** (“trcoff Subroutine” on page 350) subroutine, **trcon** (“trcon Subroutine” on page 351) subroutine, **trcstart** (“trcstart Subroutine” on page 352) subroutine, **trcstop** (“trcstop Subroutine” on page 353) subroutine.

The **trace** daemon.

The **trcgenk** kernel service, **trcgenkt** kernel service.

trchook or utrchook Subroutine

Purpose

Records a trace event.

Library

Runtime Services Library (**librts.a**)

Syntax

```
#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;
```

Description

The **trchhook** 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 **utrchhook** subroutine uses a special FAST-SVC path to improve performance and is intended for programs running at user (application) level.

Parameters

d1, d2, d3, d4, d5
HkWord

Up to 5 words of data from the calling program.

An unsigned integer 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **trcgen** (“trcgen or trcgent Subroutine” on page 348) subroutine, **trcgent** (“trcgen or trcgent Subroutine” on page 348) subroutine, **trcoff** (“trcoff Subroutine”) subroutine, **trcon** (“trcon Subroutine” on page 351) subroutine, **trcstart** (“trcstart Subroutine” on page 352) subroutine, **trcstop** (“trcstop Subroutine” on page 353) 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

```
int trcoff( Channel)
int Channel;
```

Description

The **trcoff** subroutine issues an **ioctl** subroutine to the trace device driver to stop trace data collection for a particular 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

0 The **ioctl** subroutine was successful. Trace data collection stops.
-1 The **ioctl** subroutine was not successful.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **ioctl** subroutine, **trcgen** (“trcgen or trcgent Subroutine” on page 348) subroutine, **trchhook** (“trchhook or utrchhook Subroutine” on page 349) subroutine, **trcon** (“trcon Subroutine”) subroutine, **trcstart** (“trcstart Subroutine” on page 352) subroutine, **trcstop** (“trcstop Subroutine” on page 353) 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

```
int trcon( Channel)
int Channel;
```

Description

The **trcon** subroutine issues an **ioctl** subroutine to the trace device driver to start trace data collection for a particular trace channel. The trace session must have already been started using the **trace** command or the **trcstart** (“trcstart Subroutine” on page 352) 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

0 The **ioctl** subroutine was successful. Trace data collection starts.
-1 The **ioctl** subroutine was not successful.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **ioctl** subroutine, **trcgen** (“trcgen or trcgent Subroutine” on page 348) subroutine, **trchook** (“trchook or utrchook Subroutine” on page 349) subroutine, **trcoff** (“trcoff Subroutine” on page 350) subroutine, **trcstart** (“trcstart Subroutine”) subroutine, **trcstop** (“trcstop Subroutine” on page 353) subroutine.

The **trace** daemon.

The **trcgenk** kernel service, **trcgenkt** kernel service.

trcstart Subroutine

Purpose

Starts a trace session.

Library

Runtime Services Library (**librts.a**)

Syntax

```
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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Files

`/dev/trace` Trace special file.

Related Information

The **trcon** (“trcon Subroutine” on page 351) subroutine.

The **trace** daemon.

trcstop Subroutine

Purpose

Stops a trace session.

Library

Runtime Services Library (**librts.a**)

Syntax

```
int trcstop( Channel)
int 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **ioctl** subroutine, **trcgen** (“trcgen or trcgent Subroutine” on page 348) subroutine, **trchook** (“trchook or utrchook Subroutine” on page 349) subroutine, **trcoff** (“trcoff Subroutine” on page 350) subroutine, **trcon** (“trcon Subroutine” on page 351) subroutine, **trcstart** (“trcstart Subroutine” on page 352) subroutine.

The **trace** daemon.

The **trcgenk** kernel service, **trcgenkt** kernel service.

truncate, truncate64, ftruncate, or ftruncate64 Subroutine

Purpose

Changes the length of regular files.

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.

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

<i>Path</i>	Specifies the name of a file that is opened, truncated, and then closed.
<i>FileDescriptor</i>	Specifies the descriptor of a file that must be open for writing.
<i>Length</i>	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.

The **truncate** 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 <i>length</i> is greater than the offset maximum established in the open file description associated with <i>files</i> .

EINVAL	The file is not a regular file.
EINVAL	The <i>Length</i> 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.
EROFS	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 **ftruncate** subroutine fails if the following is true:

EBADF	The <i>FileDescriptor</i> parameter is not a valid file descriptor open for writing.
EINVAL	The <i>FileDescriptor</i> argument references a file that was opened without write permission.

The **truncate** function will fail if:

EACCES	A component of the path prefix denies search permission, or write permission is denied on the file.
EISDIR	The named file is a directory.
ELOOP	Too many symbolic links were encountered in resolving <i>path</i> .
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 <i>path</i> does not name an existing file or <i>path</i> is an empty string.
ENTDIR	A component of the path prefix of <i>path</i> 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 .
---------------------	---

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **fclear** subroutine, **openx**, **open**, or **creat** subroutine.

"Appendix A. Base Operating System Error Codes for Services That Require Path-Name Resolution" on page 665.

Files, Directories, and File Systems for Programmers in *AIX 5L Version 5.1 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

```
#include <search.h>
```

```
void *tsearch ( Key, RootPointer, ComparisonPointer)
const void *Key;
void **RootPointer;
int (*ComparisonPointer) (const void *Element1, const void *Element2);

void *tdelete (Key, RootPointer, ComparisonPointer)
const void *Key;
void **RootPointer;
int (*ComparisonPointer) (const void *Element1, const void *Element2);

void *tfind (Key, RootPointer, ComparisonPointer)
const void *Key;
void *const *RootPointer;
int (*ComparisonPointer) (const void *Element1, const 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 the null value 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

<i>Key</i>	Points to the data to be located.
<i>ComparisonPointer</i>	Points to the comparison function, which is called with two parameters that point to the elements being compared.
<i>RootPointer</i>	Points to a variable that in turn points to the root of the tree.
<i>Action</i>	Names a routine to be invoked at each node.
<i>Root</i>	Points to the roots of a binary search node.

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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **bsearch** subroutine, **hsearch** subroutine, **lsearch** subroutine.

ttylock, ttywait, ttyunlock, or ttylocked Subroutine

Purpose

Controls tty locking functions.

Library

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

Syntax

```
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:

```
rc = ttylock("tty0");
```

2. To lock **/dev/tty0** device and wait for lock to be cleared if it exists, use the following statements:

```
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 **tylock** subroutine, use the following statement:

```
ttyunlock("tty0");
```

4. To check for a lock on **/dev/tty0**, use the following statement:

```
rc = ttylocked("tty0");
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **/etc/locks** directory.

The Input and Output Handling Programmer's Overview in *AIX 5L Version 5.1 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

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

```
char *ttyname( FileDescriptor)
```

```
int FileDescriptor;
```

```
int isatty(FileDescriptor)
```

```
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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Files

*/dev/** Terminal device special files.

Related Information

The **ttyslot** (“ttyslot Subroutine”) subroutine.

The Input and Output Handling Programmer’s Overview in *AIX 5L Version 5.1 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **getutent** subroutine, **ttyname** or **isatty** (“**ttyname** or **isatty** Subroutine” on page 360) subroutine.

The Input and Output Handling Programmer's Overview in *AIX 5L Version 5.1 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:

```
#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:

```
#include <ulimit.h>
```

```
long int ulimit (Command, NewLimit)
int Command;
int 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.

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:

```
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**.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **brk** subroutine, **sbrk** subroutine, **getrlimit** or **setrlimit** subroutine, **pathconf** subroutine, **read** (“read, readx, readv, readvx, or pread Subroutine” on page 9) subroutines, **vlimit** subroutine, **write** (“write, writex, writev, writevx or pwrite Subroutines” on page 449) 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **chmod** subroutine, **mkdir** subroutine, **mkfifo** subroutine, **mknod** subroutine, **openx**, **open**, or **creat** subroutine, **stat** (“statx, stat, lstat, fstatx, fstat, fullstat, ffullstat, stat64, lstat64, or fstat64 Subroutine” on page 251) subroutine.

The **sh** command, **ksh** command.

The **sys/mode.h** file.

Shells Overview in *AIX 5L Version 5.1 System User's Guide: Operating System and Devices*.

Files, Directories, and File Systems for Programmers in *AIX 5L Version 5.1 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 unmount a device, file and directory mount.

Parameters

<i>Device</i>	The path name of the block device to be unmounted for the uvmount subroutine.
<i>VirtualFileSystemID</i>	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 <i>VirtualFileSystemID</i> is also reported in the stat subroutine <i>st_vfs</i> field.
<i>Flag</i>	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 <i>VirtualFileSystemID</i> 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 <i>Device</i> parameter does not exist.
ENOBLK	The <i>Device</i> parameter is not a block device.
EINVAL	The <i>Device</i> parameter is not mounted.
EINVAL	The <i>Device</i> 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."

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **mount** (“vmount or mount Subroutine” on page 380) subroutine.

The **mount** command, **umount** command.

Mounting Overview in *AIX 5L Version 5.1 System Management Concepts: Operating System and Devices*.

Files, Directories, and File Systems for Programmers in *AIX 5L Version 5.1 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 ( Name)
struct utsname *Name;
int unamex ( Name)
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];
```

On Itanium-based platforms, the **utsname** structure contains the additional member:

```
char  procserial[SYS_NMLM];
```

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 **unamex** subroutine uses the **xutsname** structure, which is defined in the **sys/utsname.h** file, and contains the following members:

```
unsigned long  nid;
long  reserved[3];
```

The `xutsname.nid` field is the binary form of the `utsname.machine` field. 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 the **utsname** 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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

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 (**fseek**, **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

<i>Character</i>	Specifies a character.
<i>Stream</i>	Specifies the input stream.

Return Values

The **ungetc** and **ungetwc** subroutines return the inserted character if successful; otherwise, **EOF** or **WEOF** is returned, respectively.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

Other wide character I/O subroutines: **fgetwc** subroutine, **fgetws** subroutine, **fputwc** subroutine, **fputws** subroutine, **getwc** subroutine, **getwchar** subroutine, **getws** subroutine, **putwc** subroutine, **putwchar** subroutine, **putws** subroutine.

Related standard I/O subroutines: **fdopen** subroutine, **fgets** subroutine, **fopen** subroutine, **fprintf** subroutine, **fputc** subroutine, **fputs** subroutine, **fread** subroutine, **freopen** subroutine, **fwrite** subroutine, **gets** subroutine, **printf** subroutine, **putc** subroutine, **putchar** subroutine, **puts** subroutine, **putw** subroutine, **sprintf** subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Wide Character Input/Output Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

unlink Subroutine

Purpose

Removes a directory entry.

Library

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

Syntax

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

```
int unlink ( Path)  
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.
EACCESS	Write permission is denied on the directory containing the link to be removed.
EPERM	The named file is a directory, and the calling process does not have root user authority.
EBUSY	The entry to be unlinked is the mount point for a mounted filesystem, or the file named by <i>Path</i> is a named STREAM.
EPERM	The file specified by the <i>Path</i> parameter is a directory, and the calling process does not have root user authority.
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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **close** subroutine, **link** subroutine, **open** subroutine, **remove** (“remove Subroutine” on page 34) subroutine, **rmdir** (“rmdir Subroutine” on page 38) subroutine.

The **rm** command.

Files, Directories, and File Systems for Programmers in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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*.

If the program calling the **unload** subroutine was linked on 4.2 or a later release, 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **load** subroutine, **loadbind** subroutine, **loadquery** subroutine, **dlclose** subroutine.

The **ld** command.

Subroutines Overview in *AIX 5L Version 5.1 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 ( FileDescriptor )  
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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **grantpt** subroutine.

The Input and Output Handling Programmer's Overview in *AIX 5L Version 5.1 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 **SETUINFO** or **GETUINFO**.
- EINVAL** The *Command* parameter is set to **SETUINFO**, and the *Count* parameter is larger than **UINFOSIZ**.
- EFAULT** The *Buffer* parameter points outside of the address space of the process.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **getuinfo** subroutine, **setpenv** (“setpenv Subroutine” on page 113) subroutine.

The **login** command.

List of Security and Auditing Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Subroutines Overview in *AIX 5L Version 5.1 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.

Parameters

Path Points to the file.

Times 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."

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Microsecond time stamps are not implemented, even though the **utimes** subroutine provides a way to specify them.

Related Information

The **stat** ("statx, stat, lstat, fstatx, fstat, fullstat, fullstat, stat64, lstat64, or fstat64 Subroutine" on page 251) subroutine.

Files, Directories, and File Systems for Programmers in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

varargs Macros

Purpose

Handles a variable-length parameter list.

Library

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

Syntax

```
#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

```
#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, **execl** 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 **execl** system call implementations are examples of the **varargs** macros usage.

1. The following example includes **<stdarg.h>**:

```
#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");
    /* ArguementV[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>**:

```
#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));
}
```

Implementation Specifics

These macros are part of Base Operating System (BOS) Runtime.

Related Information

The **exec** subroutines.

The **printf** subroutine.

List of String Manipulation Services in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

vfwprintf, 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**, **fwprintf** 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 **fwprintf**.

Error Codes

Refer to **fwprintf**.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) subroutine.

Related Information

The **fwprintf** subroutine.

The **wchar.h** file.

vmgetinfo Subroutine

Purpose

Retrieves Virtual Memory Manager 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 Virtual Memory Manager parameters.

Implementation Specifics

This subroutine is part of the Base Operating System (BOS) Runtime.

Parameters

<i>arg</i>	Additional parameter which depends on the command parameter.
<i>command</i>	Specifies which information should be returned. The <i>command</i> parameter has the following valid value: <i>VMINFO</i> The content of <i>vminfo</i> structure (described in <i>sys/vminfo.h</i>) is returned. <i>out</i> should point to a struct <i>vminfo</i> and <i>arg</i> should be the size of this structure. The smaller of <i>arg</i> or <i>sizeof</i> (struct <i>vminfo</i>) will be copied.
<i>out</i>	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:

<i>EFAULT</i>	The copy operation to the buffer was not successful.
<i>ENOSYS</i>	The command parameter is not valid (or not yet implemented).

vmount or mount Subroutine

Purpose

Makes a file system available for use.

Library

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

Syntax

```
#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_vfsnumber

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 <i>VMount</i> 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 <i>VMount</i> parameter are unintelligible (for example, the <code>vmt_gfstype</code> 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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **mntctl** subroutine, **umount** (“umount or uvmount Subroutine” on page 365) subroutine.

The **mount** command, **umount** command.

Files, Directories, and File Systems for Programmers in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

vsnprintf Subroutine

Purpose

Print formatted output.

Library

Standard library (**libc.a**)

Syntax

```
#include <stdarg.h>
#include <stdio.h>
int vsnprintf(char * s, size_t n, const char * format, va_list ap)
```

Description

Refer to **fprintf**.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) subroutine.

vwsprintf 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

<i>wcs</i>	Specifies the array of wide characters where the output is to be written.
<i>Format</i>	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.
<i>arg</i>	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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **vsprintf** subroutine.

The **printf** command.

National Language Support Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

wait, waitpid, wait3, or wait364 Subroutine

Purpose

Waits for a child process to stop or terminate.

Library

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

Syntax

```
#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** ("sigaction, sigvec, or signal Subroutine" on page 135) 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_SEWTED** 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

<i>StatusLocation</i>	Points to integer variable that contains (or will contain) the child process termination status, as defined in the sys/wait.h file.
<i>ProcessID</i>	Specifies the child process.
<i>Options</i>	Modifies behavior of subroutine.
<i>ResourceUsage</i>	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*)
pid_t *ReturnedValue*;

Returns a nonzero value if status returned for a child process that has continued from a job control stop.

WIFSTOPPED(*ReturnedValue*)
int *ReturnedValue*;

Returns a nonzero value if status returned for a stopped child.

int
WSTOPSIG(*ReturnedValue*)
int *ReturnedValue*;

Returns the number of the signal that caused the child to stop.

WIFEXITED(*ReturnedValue*)
int *ReturnedValue*;

Returns a nonzero value if status returned for normal termination.

int
WEXITSTATUS(*ReturnedValue*)
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.
EINTR	This subroutine was terminated by receipt of a signal.
EFAULT	The <i>StatusLocation</i> or <i>ResourceUsage</i> parameter points to a location outside of the address space of the process.

The **waitpid** subroutine is unsuccessful if the following is true:

ECHILD	The process or process group ID specified by the <i>ProcessID</i> parameter does not exist or is not a child process of the calling process.
---------------	--

The **waitpid** and **wait3** subroutines are unsuccessful if the following is true:

EINVAL	The value of the <i>Options</i> parameter is not valid.
---------------	---

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **exec** subroutine, **_exit**, **exit**, or **atexit** subroutine, **fork** subroutine, **getrusage** subroutine, **pause** subroutine, **ptrace** subroutine, **sigaction** ("sigaction, sigvec, or signal Subroutine" on page 135) subroutine.

wcscat, wcschr, wcscmp, wcsncpy, or wcsncpy Subroutine

Purpose

Performs operations on wide-character strings.

Library

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

Syntax

```
#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 * wcsncpy(WcString1, WcString2)
wchar_t *WcString1;
const wchar_t
*
WcString2;
```

```
size_t wcsncpy(WcString1, WcString2)
const wchar_t *WcString1, *WcString2;
```

Description

The **wcscat**, **wcschr**, **wcscmp**, **wcsncpy**, or **wcsncpy** 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 **wcsncpy** 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 **wcsncpy** subroutine returns the number of **wchar_t** characters in the segment.

Parameters

<i>WcString1</i>	Points to a wide-character string.
<i>WcString2</i>	Points to a wide-character string.
<i>WideCharacter</i>	Specifies a wide character for location.

Return Values

Upon successful completion, the **wcscat** and **wcscpy** 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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **mbscat** subroutine, **mbchr** subroutine, **mbscmp** subroutine, **mbscopy** subroutine, **mbsrchr** subroutine, **wcsncat** (“*wcsncat*, *wcsncmp*, or *wcsncpy* Subroutine” on page 394) subroutine, **wcsncmp** (“*wcsncat*, *wcsncmp*, or *wcsncpy* Subroutine” on page 394) subroutine, **wcsncpy** (“*wcsncat*, *wcsncmp*, or *wcsncpy* Subroutine” on page 394) subroutine, **wcsrchr** (“*wcsrchr* Subroutine” on page 396) subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Wide Character String Comparison Subroutines, Understanding Wide Character String Copy Subroutines, Understanding Wide Character String Search Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

wcscoll Subroutine

Purpose

Compares wide character strings.

Library

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

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 **wscoll** 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 **wscoll** 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 **wscoll** 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **wcscmp** (“**wcscat**, **wcschr**, **wscmp**, **wscopy**, or **wcscspn** Subroutine” on page 388) subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Wide Character String Collation Subroutines in *AIX 5L Version 5.1 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

<i>WcString</i>	Contains the output of the wcsftime subroutine.
<i>Maxsize</i>	Specifies the maximum number of bytes (including the wide character null-terminating byte) that may be placed in the <i>WcString</i> parameter.
<i>Format</i>	Specifiers are the same as in strftime (“strftime Subroutine” on page 262) function.
<i>TimPtr</i>	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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **mbstowcs** subroutine, **strfmon** (“strfmon Subroutine” on page 260) subroutine, **strftime** (“strftime Subroutine” on page 262) subroutine, **strptime** (“strptime Subroutine” on page 271) subroutine.

National Language Support Overview for Programming, Understanding Time and Monetary Formatting Subroutines, Subroutines Overview in *AIX 5L Version 5.1 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 ( WC)
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 5.1 System Management Concepts: Operating System and Devices* for more information.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **csid** subroutine, **mbstowcs** subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Multibyte Code and Wide Character Code Conversion Subroutines, Understanding the Character Set Description (charmap) Source File in *AIX 5L Version 5.1 System Management Concepts: Operating System and Devices*.

wcslen Subroutine

Purpose

Determines the number of characters in a wide-character string.

Library

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

Syntax

```
#include <wctype.h>
```

```
size_t wcslen( WcString)
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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **mbslen** subroutine, **wctomb** (“wctomb Subroutine” on page 412) subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Multibyte Code and Wide Character Code Conversion Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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 * wcsncat (WcString1, WcString2, Number)
wchar_t * WcString1;
const wchar_t * WcString2;
size_t Number;
```

```
wchar_t * wcsncmp (WcString1, WcString2, Number)
const wchar_t *WcString1, *WcString2;
size_t Number;
```

```
wchar_t * wcsncpy (WcString1, WcString2, Number)
wchar_t *WcString1;
const wchar_t *WcString2;
size_t Number;
```

Description

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

<i>WcString1</i>	Specifies a wide-character string.
<i>WcString2</i>	Specifies a wide-character string.
<i>Number</i>	Specifies the range of characters to process.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **mbsncat** subroutine, **mbsncmp** subroutine, **mbsncpy** subroutine, **wcscat** (“wcscat, wcschr, wcsncmp, wcsncpy, or wcsncpy Subroutine” on page 388) subroutine, **wcscmp** (“wcscat, wcschr, wcsncmp, wcsncpy, or wcsncpy Subroutine” on page 388) subroutine, **wcscpy** (“wcscat, wcschr, wcsncmp, wcsncpy, or wcsncpy Subroutine” on page 388) subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Wide Character String Comparison Subroutines, Understanding Wide Character String Copy Subroutines in *AIX 5L Version 5.1 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

```
#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

<i>WcString1</i>	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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **mbspbrk** subroutine, **wcschr** (“wcscat, wcschr, wcscmp, wcscopy, or wcsncpy Subroutine” on page 388) subroutine, **wcscspn** (“wcscat, wcschr, wcscmp, wcscopy, or wcsncpy Subroutine” on page 388) subroutine, **wcsrchr** (“wcsrchr Subroutine”) subroutine, **wcsspn** (“wcsspncpy Subroutine” on page 398) subroutine, **wcstok** (“wcstok Subroutine” on page 401) subroutine, **wcswcs** (“wcswcs Subroutine” on page 408) subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Wide Character String Search Subroutines in *AIX 5L Version 5.1 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

```
#include <wctr.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

<i>WcString</i>	Points to a string.
<i>WideCharacter</i>	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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **mbschr** subroutine, **mbsrchr** subroutine, **wcschr** (“wscat, wcschr, wscmp, wscopy, or wcsdspn Subroutine” on page 388) subroutine, **wcscspn** (“wscat, wcschr, wscmp, wscopy, or wcsdspn Subroutine” on page 388) subroutine, **wcspbrk** (“wcspbrk Subroutine” on page 395) subroutine, **wcsspn** (“wcsspn Subroutine” on page 398) subroutine, **wcstok** (“wcstok Subroutine” on page 401) subroutine, **wcswcs** (“wcswcs Subroutine” on page 408) subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Wide Character String Search Subroutines in *AIX 5L Version 5.1 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

```
#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 **wcrtomb** function.

If **dst** is not a null pointer, the pointer object pointed to by **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 **ps** is a null pointer, the **wcsrtombs** function uses its own internal **mbstate_t** object, which is initialised at program startup to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by **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 **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 **wcsrtombs** function stores the value of the macro **EILSEQ** in **errno** and returns **(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 **wcsrtombs** function may fail if:

EINVAL **ps** points to an object that contains an invalid conversion state.
EILSEQ A wide-character code does not correspond to a valid character.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) subroutine.

Related Information

The **wctomb** (“wctomb Subroutine” on page 412) subroutine.

The **wchar.h** file.

wcsspn Subroutine

Purpose

Returns the number of wide characters in the initial segment of a string.

Library

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

Syntax

```
#include <wctype.h>
```

```
size_t wcsspn( WcString1, WcString2)  
const wchar_t *WcString1, *WcString2;
```

Description

The **wcsspn** subroutine computes the number of **wchar_t** characters in the initial segment of the string pointed to by the *WcString1* parameter. The *WcString1* parameter consists entirely of **wchar_t** characters from the string pointed to by the *WcString2* parameter.

Parameters

WcString1 Points to the initial segment of a string.
WcString2 Points to a set of characters string.

Return Values

The **wcsspn** subroutine returns the number of **wchar_t** characters in the segment.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **wcschr** (“wscat, wcschr, wscmp, wscpy, or wscspn Subroutine” on page 388) subroutine, **wcscspn** (“wscat, wcschr, wscmp, wscpy, or wscspn Subroutine” on page 388) subroutine, **wcspbrk** (“wcpbrk Subroutine” on page 395) subroutine, **wcsrchr** (“wcsrchr Subroutine” on page 396) subroutine, **wcstok** (“wcstok Subroutine” on page 401) subroutine, **wcswcs** (“wcswcs Subroutine” on page 408) subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Wide Character String Search Subroutines in *AIX 5L Version 5.1 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**.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) subroutine.

Related Information

The **wchar.h** file.

wcstod Subroutine

Purpose

Converts a wide character string to a double-precision number.

Library

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

Syntax

```
#include <stdlib.h>
```

```
double wcstod ( Nptr, Endptr)  
const wchar_t *Nptr;  
wchar_t **Endptr;
```

Description

The **wcstod** subroutine converts the initial portion of the wide character string pointed to by the *Nptr* parameter to a double-precision number. The input wide character string is first broken down into three parts:

1. An initial, possibly empty, sequence of white-space wide character codes (as specified by the **iswspace** subroutine)
2. A subject sequence interpreted as a floating-point constant
3. A final wide character string of one or more unrecognized wide character codes (including the terminating wide character null)

The subject sequence is then (if possible) converted to a floating-point number and returned as the result of the **wcstod** subroutine.

The subject sequence is expected to consist of an optional + (plus sign) or - (minus sign), a non-empty sequence of digits (which may contain a radix), and an optional exponent. The exponent consists of e or E, followed by an optional sign, followed by one or more decimal digits. The subject sequence is the longest initial sub-sequence of the input wide character string (starting with the first non-white-space wide character code) that is of the expected form. The subject sequence contains no wide character codes if the input wide character string is empty or consists entirely of white-space wide character codes, or if the first non-white-space wide character code is other than a sign, a digit, or a radix.

If the subject sequence is valid, the sequence of wide character codes starting with the first digit or radix (whichever occurs first) is interpreted as a float or double constant. If the radix is used in place of a period, and if either an exponent or radix does not appear, a radix is assumed to follow the last digit in the wide character string. If the subject sequence begins with a - (minus sign), the conversion value is negated. A pointer to the final wide character string is stored in the object pointed to by the *Endptr* parameter, unless it specifies a null pointer. The radix is defined by the **LC_NUMERIC** category. In the C locale, or in a locale where the radix is not defined, the radix defaults to a period.

Parameters

Nptr Contains a pointer to the wide character string to be converted to a double-precision value.
Endptr Contains a pointer to the position in the string specified by the *Nptr* parameter where a wide character is found that is not a valid character for the purpose of this conversion.

Return Values

The **wcstod** subroutine returns a converted double-precision value if a valid floating-point constant is found. If no conversion could be performed, a value of 0 is returned. If the converted value is outside the range (either too high or too low), the **errno** global variable is set to **ERANGE**. In case of overflow, plus or minus **HUGE_VAL** is returned. In the case of underflow, a value of 0 is returned. If the subject sequence is empty or does not have the expected form, no conversion is performed. In this case, the value specified by the *Nptr* parameter is stored in the object pointed to by the *Endptr* parameter, provided that the *Endptr* parameter is not a null pointer.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **iswspace** subroutine, **wcstol** (“wcstol or wcstoll Subroutine” on page 403) subroutine, **wcstoul** (“wcstoul or wcstoull Subroutine” on page 406) subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Wide Character String Conversion Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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 null pointer 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

<i>ptr</i>	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.
<i>WcString1</i>	Contains a pointer to the wide-character string to be searched.
<i>WcString2</i>	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"*/
}
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **wcschr** (“**wcscat**, **wcschr**, **wcscmp**, **wcscpy**, or **wcscspn** Subroutine” on page 388) subroutine, **wcscspn** (“**wcscat**, **wcschr**, **wcscmp**, **wcscpy**, or **wcscspn** Subroutine” on page 388) subroutine, **wcspbrk** (“**wcspbrk** Subroutine” on page 395) subroutine, **wcsrchr** (“**wcsrchr** Subroutine” on page 396) subroutine, **wcsspn** (“**wcsspn** Subroutine” on page 398) subroutine, **wcstod** (“**wcstod** Subroutine” on page 399) subroutine, **wcstol** (“**wcstol** or **wcstoll** Subroutine” on page 403) subroutine, **wcstoul** (“**wcstoul** or **wcstoull** Subroutine” on page 406) subroutine, **wcswcs** (“**wcswcs** Subroutine” on page 408) subroutine.

The **wchar.h** file.

National Language Support Overview for Programming, Subroutines Overview, Understanding Wide Character String Search Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

wcstol or wcstoll Subroutine

Purpose

Converts a wide-character string to a long integer representation.

Library

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

Syntax

```
#include <stdlib.h>
```

```
long int wcstol ( Nptr, Endptr, Base)
const wchar_t *Nptr;
wchar_t **Endptr;
int Base;

long long int wcstoll (*Nptr, **Endptr, Base)
const wchar_t *Nptr;
wchar_t **Endptr;
int Base
```

Description

The **wcstol** subroutine converts a wide-character string to a long integer representation. The **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 **iswspace** subroutine)
2. A subject sequence interpreted as an integer and represented in a radix determined by the *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 *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

<i>Nptr</i>	Contains a pointer to the wide-character string to be converted to a long integer number.
<i>Endptr</i>	Contains a pointer to the position in the <i>Nptr</i> parameter string where a wide-character is found that is not a valid character.
<i>Base</i>	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 (according to the sign of the value), and 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** 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 then 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:

```
#include <stdlib.h>
#include <locale.h>
#include <errno.h>

main()
{
    wchar_t *WCString, *endptr;
    long int retval;
    (void)setlocale(LC_ALL, "");
    /**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 */
}
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **iswspace** subroutine, **wcstod** (“wcstod Subroutine” on page 399) subroutine, **wcstoul** (“wcstoul or wcstoull Subroutine” on page 406) subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Wide Character String Conversion Subroutines in *AIX 5L Version 5.1 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

<i>String</i>	Points to the area where the result of the conversion is stored. If the <i>String</i> parameter is a null pointer, the subroutine returns the number of bytes required to hold the conversion.
<i>WcString</i>	Points to a wide-character string.
<i>Number</i>	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.
---------------	---

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **mbstowcs** subroutine, **mbtowc** subroutine, **wcslen** (“wcslen Subroutine” on page 393) subroutine, **wctomb** (“wctomb Subroutine” on page 412) subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Multibyte Code and Wide Character Code Conversion Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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 *Endptr* parameter to the position where an unrecognized character, including the terminating null, is found.

The base specified by the *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 *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 **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 *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 *Endptr* parameter on return from the call to this subroutine.

Parameters

<i>Nptr</i>	Contains a pointer to the wide character string to be converted to an unsigned long integer.
<i>Endptr</i>	Contains a pointer to the position in the <i>Nptr</i> string where a wide character is found that is not a valid character for the purpose of this conversion.
<i>Base</i>	Specifies the radix in which the wide characters are interpreted.

Return Values

The **wcstoul** and **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 **ULONGLONG_MAX** value is returned, and the value of the **errno** global variable is set to a **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 *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 *Nptr* parameter is stored in the *Endptr* parameter if it is not a null pointer.

If the radix specified by the *Base* parameter is not supported, an **EINVAL** value is returned. If the value to be returned is not representable, an **ERANGE** value is returned.

Examples

To convert a wide character string to an unsigned long integer, use the following code:

```
#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. */
}
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

National Language Support Overview for Programming, Subroutines Overview, Understanding Wide Character String Conversion Subroutines in *AIX 5L Version 5.1 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

<i>WcString1</i>	Points to the wide-character string being searched.
<i>WcString2</i>	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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **mbspbrk** subroutine, **wcschr** (“wscat, wcschr, wscmp, wcsncpy, or wcsncpy Subroutine” on page 388) subroutine, **wcscspn** (“wscat, wcschr, wscmp, wcsncpy, or wcsncpy Subroutine” on page 388) subroutine, **wcspbrk** (“wcspbrk Subroutine” on page 395) subroutine, **wcsrchr** (“wcsrchr Subroutine” on page 396) subroutine, **wcsspn** (“wcsspncpy Subroutine” on page 398) subroutine, **wcstok** (“wcstok Subroutine” on page 401) subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Wide Character String Search Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

wcswidth Subroutine

Purpose

Determines the display width of wide character strings.

Library

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

Syntax

```
#include <string.h>
```

```
int wcswidth (* Pwcs, n)
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:

```
#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.
        */
    }
}

```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **wcwidth** (“wcwidth Subroutine” on page 415) subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Wide Character Display Column Width Subroutines in *AIX 5L Version 5.1 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

```
#include <string.h>
```

```

size_t wcsxfrm ( WcString1, WcString2, Number)
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 **wscmp** subroutine, the result is the same as that obtained by a direct call to the **wscoll** subroutine on the two original wide-character strings.

Parameters

<i>WcString1</i>	Points to the destination wide-character string.
<i>WcString2</i>	Points to the source wide-character string.
<i>Number</i>	Specifies the maximum number of wide-character codes to place into the array specified by <i>WcString1</i> . To determine the necessary size specification, set the <i>Number</i> parameter to a value of 0, so that the <i>WcString1</i> 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**.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **wcscmp** (“*wcscat*, *wcschr*, *wcscmp*, *wcscpy*, or *wcscspn* Subroutine” on page 388) subroutine, **wcscoll** (“*wcscoll* Subroutine” on page 390) subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Wide Character String Collation Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

wctob Subroutine

Purpose

Wide-character to single-byte conversion.

Library

Standard library (**libc.a**)

Syntax

```
#include <stdio.h>
#include <wchar.h>
int wctob (wint_t c);
```

Description

The **wctob** function determines whether **c** corresponds to a member of the extended character set whose character representation is a single byte when in the initial shift state.

The behavior of this function is affected by the **LC_CTYPE** category of the current locale.

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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) subroutine.

Related Information

The **btowc** subroutine.

The **wchar.h** file.

wctomb Subroutine

Purpose

Converts a wide character into a multibyte character.

Library

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

Syntax

```
#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

<i>Storage</i>	Points to an area where the result of the conversion is stored.
<i>WideCharacter</i>	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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **mbtowc** subroutine, **mbstowcs** subroutine, **wcslen** (“wcslen Subroutine” on page 393) subroutine, **wcstombs** (“wcstombs Subroutine” on page 405) subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Multibyte Code and Wide Character Code Conversion Subroutines in *AIX 5L Version 5.1 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) subroutine.

Related Information

The **towctrans** ("towctrans Subroutine" on page 332) subroutine.

The **wctype.h** file.

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 ( Property)  
const char *Property;
```

```
wctype_t get_wctype ( Property)  
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.

Parameters

<i>Property</i>	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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

The **wctype** subroutine adheres to *Systems Interface and Headers, Issue 4 of X/Open*.

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** (“setlocale Subroutine” on page 107) subroutine, **towlower** (“towlower Subroutine” on page 333) subroutine, **towupper** (“towupper Subroutine” on page 334) subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Wide Character Classification Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

wcwidth Subroutine

Purpose

Determines the display width of wide characters.

Library

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

Syntax

```
#include <string.h>
```

```
int wcwidth ( WC)
```

```
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:

```
#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.
        */
    }
}

```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **wcswidth** (“wcswidth Subroutine” on page 409) subroutine.

National Language Support Overview for Programming, Subroutines Overview, Understanding Wide Character Display Column Width Subroutines in *AIX 5L Version 5.1 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

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

```
int wlm_assign ( args)
```

```
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 *AIX 5L Version 5.1 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.

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_pgids Specifies the address of the array containing the process group identifiers (pids) 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_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

Manual Assignment and Workload Manager Application Programming Interface in *AIX 5L Version 5.1 System Management Concepts: Operating System and Devices*.

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 ('') 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 **structclass_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 5.1 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.

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
confdir

Needs to be initialized with **WLM_VERSION**. 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 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.

The **wlm_create_class** (“**wlm_create_class** Subroutine” on page 423) subroutine, **wlm_delete_class** (“**wlm_delete_class** Subroutine” on page 425) subroutine.

Workload Manager Application Programming Interface in *AIX 5L Version 5.1 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**.

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_checks** 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.

System Management Concepts: Operating System and Devices, Chapter 13 Workload Manager, **Automatic class Assignment**.

The rules file.

wlm_classify Subroutine

Purpose

Determines which classes a process is assigned to.

Library

Workload Manager Library (**libwlm.a**)

Syntax

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

int wlm_classify ( config, attributes, class, len)

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 (!), 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/dbserve - _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

<i>config</i>	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 <i>current</i> as the default configuration.
<i>attributes</i>	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.
<i>class</i>	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.
<i>len</i>	Specifies a pointer to an integer containing the length in bytes of the buffer pointed to by <i>class</i> when calling wlm_classify and the actual number of class names copied into the <i>class</i> 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 5.1 Files Reference*.

Automatic assignment (“wlm_classify Subroutine” on page 421) in *AIX 5L Version 5.1 System Management Concepts: Operating System and Devices*.

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:

<i>WLM_NOT_INITED</i>	Missing call to wlm_init .
<i>WLM_EFAULT</i>	Invalid key or args pointer.
<i>WLM_BADCNAME</i>	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 **structwlm_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 ('') 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 5.1 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.

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
confdir

Needs to be initialized with **WLM_VERSION**.

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 '').

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 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** (“wlm_change_class Subroutine” on page 418) subroutine, **wlm_delete_class** (“wlm_delete_class Subroutine”) subroutine.

Workload Management in *AIX 5L Version 5.1 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.

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
confdir

Needs to be initialized with **WLM_VERSION**.

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** (“wlm_change_class Subroutine” on page 418) subroutine, **wlm_create_class** (“wlm_create_class Subroutine” on page 423) subroutine.

Workload Management in *AIX 5L Version 5.1 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

```
#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:

<i>WLM_BADVERS</i>	Bad version number.
<i>WLM_NOT_INITED</i>	Missing call to wlm_init .
<i>WLM_EFAULT</i>	Invalid <i>ctx</i> or <i>args</i> 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

```
#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'd 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.

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 **wlm.h** header file in the *AIX 5L Version 5.1 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), **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 **wlm_get_info** subroutine does not require any special privileges and is accessible to all users. **wlm_get_info** fails if WLM is off.

Parameters

wlmargs

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**. Optionally, the following flag value 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_VERBOSE_MODE

Shows the system-defined subclasses (Default and Shared) even if they have not been modified by a WLM administrator.

WLM_SUPER_ONLY and **WLM_SUB_ONLY** are mutually exclusive.

name Contains either a null string or the name of a valid superclass or subclass (in the form **Super.Sub**). This field can be used in conjunction with the flags to further narrow the scope of **wlm_get_info**:

- If the name of a subclass is provided, **wlm_get_info** returns the statistics only for the specified subclass.
- If the name of a superclass is provided or if none of the **WLM_SUPER_ONLY** and **WLM_SUB_ONLY** flag is provided, **wlm_get_info** returns the statistics for the specified superclass and all its subclasses.
- If the name of a superclass is provided together with **WLM_SUPER_ONLY**, **wlm_get_info** returns only the statistics for the specified superclass.
- If the name of a superclass is provided together with **WLM_SUB_ONLY**, **wlm_get_info** returns the statistics for all the subclasses of the specified superclass.

All the other fields of the **wlm_args** structure can be left uninitialized.

info The address of an array of structures of type **struct wlm_info**. Upon successful return from **wlm_get_info**, this array contains the WLM statistics for the classes selected.

count The address of an integer containing the maximum number of element (of type **wlm_info**) for **wlm_get_info** to copy into the array above. If the call to **wlm_get_info** is successful, this integer

contains the number of elements actually copied. If the initial value is equal to zero (0), **wlm_get_info** sets this value to the number of classes selected by the specified combination of **versflags** and **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_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** (“wlm_change_class Subroutine” on page 418) subroutine, **wlm_create_class** (“wlm_create_class Subroutine” on page 423) subroutine, **wlm_delete_class** (“wlm_delete_class Subroutine” on page 425) subroutine.

wlm_initialize Subroutine

Purpose

Prepares Workload Manager (WLM) for use by an application.

Library

Workload Manager Library (**libwlm.a**)

Syntax

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

```
int wlm_initialize ( 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 <i>flags</i> 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

```
#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:

<i>WLM_BADVERS</i>	Bad version number.
<i>WLM_NOT_INITED</i>	Missing call to wlm_init .
<i>WLM_NOMEM</i>	Not enough memory.
<i>WLM_NOCLASS</i>	Specified configuration does not exist.
<i>WLM_EFAULT</i>	Invalid ctx or args argument.

Related Information

The **wlm_endkey** subroutine.

The **wlm_class2key** subroutine.

The **wlm_key2class** subroutine.

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.
- *ctx* The context handler returned by **wlm_initkey**, or a NULL pointer otherwise. .
- *key* The search key.

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:

<i>WLM_BADVERS</i>	Bad version number.
<i>WLM_NOT_INITED</i>	Missing call to wlm_init .
<i>WLM_NOMEM</i>	Not enough memory.
<i>WLM_NOCLASS</i>	No class matching the key was found.
<i>WLM_EFAULT</i>	Invalid <i>ctx</i> or <i>args</i> 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

```
#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. If no superclass name is given in the **name** field of the **class_definition** substructure, the routine loads the class properties for all the superclasses of the target configuration. If a superclass name is given, then only the subclasses of the given superclass are refreshed. Flags passed in the flags portion of the **versflags** field can be used to modify the mode of operation of WLM. The values are identical to the flags values passed to the **wlm_set** API routine. Not all combinations of parameters are allowed, and different combinations may require a different level of privilege as explained below:

- To start or update WLM, the name of a configuration **must** be passed in the **confdir** field. **wlm_load** updates or starts WLM using the properties files from the given configuration. Only root can specify the name of a configuration different from the currently active configuration (specified as *current* in **confdir**).
- When WLM is on (the operation is an update) and the name of the configuration passed in the **confdir** field of the **wlm_args** structure is the name of the currently active configuration, the name of a superclass can be given in the **name** field to update only the subclasses of the given superclass. This functionality is accessible to root and to users with administration privileges on the subclasses of the superclass. The **wlm_load** subroutine cannot be used in this context to alter the state of WLM (start, stop, or switch between active and passive modes).
- If the caller of **wlm_load** has root privileges and does not specify a superclass, the flags passed in **versflags** can be used to alter WLM's mode of operation: start WLM in active or passive mode, switch between active and passive modes, or enable/disable the rset bindings.

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 can be provided:

versflags

Needs to be initialized with **WLM_VERSION**. Optionally, some of the flags used when calling **wlm_set** to change the mode of operation of WLM can be given by the root user. The valid values are **WLM_ACTIVE**, **WLM_PASSIVE**, and **WLM_BIND_RSETS**. Of course, **WLM_ACTIVE** and **WLM_PASSIVE** are mutually exclusive.

confdir

Specifies the name of the WLM configuration to be loaded into the kernel. It must be either the name of a valid subdirectory of **/etc/wlm** or the string *current* to refer to the active configuration.

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 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 `wlmcntrl` command.

The `wlm_set` (“`wlm_set Subroutine`” on page 440) 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

```
#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 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 32 when reading superclasses and 12 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 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 **lsclass** command.

The **wlm.h** header file.

wlm_set Subroutine

Purpose

Loads a Workload Manager (WLM) configuration into the kernel.

Library

Workload Manager Library (**libwlm.a**)

Syntax

```
#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 **wlmargs** 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 **passive** to **active** mode.

WLM_PASSIVE

Turns WLM on in **passive** mode or transitions from **active** to **passive** mode.

WLM_BIND_RSETS

Requests that WLM takes the resource set bindings into account.

All combinations of the flags above are not legal:

- **WLM_OFF**, **WLM_ACTIVE**, and **WLM_PASSIVE** are mutually exclusive.
- **WLM_BIND_RSETS** is ineffective when used together with **WLM_OFF**.
- Only **WLM_TEST_ON** is allowed to non-root users.

Return Values

Upon successful completion, the `wlm_set` subroutine returns a value of 0, and the current state of WLM is returned in the integer pointed to by `flags`. The return value is `WLM_OFF`, `WLM_ACTIVE`, or `WLM_PASSIVE`. When WLM is on in either active or passive mode, the `WLM_BIND_RSETS` flag is added when WLM uses resource sets bindings.

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.h` header file.

The `wlm_load` (“`wlm_load Subroutine`” on page 436) subroutine.

wlm_set_tag Subroutine

Purpose

Sets the current process's tag and related flags.

Library

Workload Manager Library (`libwlm.a`)

Syntax

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

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

```
int wlm_set_tag ( tag, flags)
```

```
char *tag;
```

```
int *flags;
```

Description

The `tag` attribute is a new attribute of a process that can be set using the Workload Manager (WLM) `wlm_set_tag` subroutine. This tag is a character string with a maximum length of `WLM_TAG_LENGTH` (not including the null terminator). Process tags can be displayed using the `ps` command.

The `tag` attribute is also one of the `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 `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 Workload Manager Application Programming Interface.

When an application sets its tag using **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 5.1 Files Reference*.

wmemchr Subroutine

Purpose

Find a wide-character in memory.

Library

Standard library (**libc.a**)

Syntax

```
#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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) subroutine.

Related Information

The **wmemcmp** (“wmemcmp Subroutine”) subroutine, **wmemcpy** (“wmemcpy Subroutine” on page 444) subroutine, **wmemmove** (“wmemmove Subroutine” on page 444) subroutine, **wmemset** (“wmemset Subroutine” on page 445) subroutine.

The **wchar.h** file.

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**.

Implementation Specifics

This subroutine is part of the Base Operating System (BOS) subroutine.

Related Information

The **wmemchr** (“wmemchr Subroutine” on page 442) subroutine, **wmemcpy** (“wmemcpy Subroutine”) subroutine, **wmemmove** (“wmemmove Subroutine”) subroutine, **wmemset** (“wmemset Subroutine” on page 445) subroutine.

The **wchar.h** file.

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 **wmemcpy** 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 **wmemcpy** function returns the value of **ws1**.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) subroutine.

Related Information

The **wmemchr** (“wmemchr Subroutine” on page 442) subroutine, **wmemcmp** (“wmemcmp Subroutine” on page 443) subroutine, **wmemmove** (“wmemmove Subroutine”) subroutine, **wmemset** (“wmemset Subroutine” on page 445) subroutine.

The **wchar.h** file.

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**.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) subroutine.

Related Information

The **wmemchr** (“wmemchr Subroutine” on page 442) subroutine, **wmemcmp** (“wmemcmp Subroutine” on page 443) subroutine, **wmemcpy** (“wmemcpy Subroutine” on page 444) subroutine, **wmemset** (“wmemset Subroutine”) subroutine.

The **wchar.h** file.

wmemset Subroutine

Purpose

Set wide-characters in memory.

Library

Standard library (**libc.a**)

Syntax

```
#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**.

Implementation Specifics

This subroutine is part of Base Operating Systems (BOS) subroutine.

Related Information

The **wmemchr** (“wmemchr Subroutine” on page 442) subroutine, **wmemcmp** (“wmemcmp Subroutine” on page 443) subroutine, **wmemcpy** (“wmemcpy Subroutine” on page 444) subroutine, **wmemmove** (“wmemmove Subroutine” on page 444) subroutine.

The **wchar.h** file.

wordexp Subroutine

Purpose

Expands tokens from a stream of words.

Library

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

Syntax

```
#include <wordexp.h>
```

```
int wordexp ( Words, Pwordexp, 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 <newline> 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 token in the list is a null pointer. The expansion of special parameters * (asterisk), @ (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

<i>Flags</i>	Contains a bit flag specifying the configurable aspects of the wordexp subroutine.
<i>Pwordexp</i>	Contains a pointer to a wordexp_t structure.
<i>Words</i>	Specifies the string containing the tokens to be expanded.

The value of the *Flags* parameter is the bitwise, inclusive OR of the constants below, which are defined in the **wordexp.h** file.

WRDE_APPEND	Appends words generated to those generated by a previous call to the wordexp subroutine.
WRDE_DOOFFS	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 <i>Pwordexp</i> parameter, it should not be set in subsequent calls to the wordexp subroutine with the <i>Pwordexp</i> parameter.
WRDE_NOCMD	Fails if command substitution is requested.
WRDE_REUSE	The <i>Pwordexp</i> parameter was passed to a previous successful call to the wordexp subroutine. Therefore, the memory previously allocated may be reused.
WRDE_SHOWERR	Does not redirect standard error to /dev/null .
WRDE_UNDEF	Reports error on an attempt to expand an undefined shell variable.

The **WRDE_APPEND** flag can be used to append a new set of words to those generated by a previous call to 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 <i>Words</i> parameter in an inappropriate context.
WRDE_BADVAL	Reference to undefined shell variable when the WRDE_UNDEF flag is set in the <i>Flags</i> parameter.
WRDE_CMDSUB	Command substitution requested when the WRDE_NOCMD flag is set in the <i>Flags</i> parameter.
WRDE_NOSPACE	Attempt to allocate memory was unsuccessful.
WRDE_SYNTAX	Shell syntax error, such as unbalanced parentheses or unterminated string.

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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **glob** subroutine, **wordfree** (“wordfree Subroutine”) 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

```
#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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **wordexp** (“wordexp Subroutine” on page 446) subroutine.

write, writex, writev, writevx or pwrite Subroutines

Purpose

Writes to a file.

Library

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

Syntax

```
#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;
int 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 **writenv** 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 **writenv** subroutine always writes a complete area before proceeding to the next.

The **writex** and **writenvx** subroutines are the same as the **write** and **writenv** 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, such as a terminal, and no data can currently be written:

- 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 **files** refers to a STREAM, the operation of **write** is determined by the values of the minimum and maximum nbyte range ("packet size") accepted by the STREAM. These values are determined by the topmost STREAM module. If nbyte falls within the packet size range, nbyte bytes will be written. If nbyte does not fall within the range and the minimum packet size value is 0, **write** will break the buffer into maximum packet size segments prior to sending the data downstream (the last segment may contain less than the maximum packet size). If nbyte does not fall within the range and the minimum value is non-zero, **write** will fail with **errno** set to **ERANGE**. Writing a zero-length buffer (nbyte is 0) to a STREAMS device sends 0 bytes with 0 returned. However, writing a zero-length buffer to a STREAMS-based pipe or FIFO sends no message and 0 is returned. The process may issue **I_SWROPT ioctl** to enable zero-length messages to be sent across the pipe or FIFO.

When writing to a STREAM, data messages are created with a priority band of 0. When writing to a STREAM that is not a pipe or FIFO:

- If **O_NONBLOCK** is clear, and the STREAM cannot accept data (the STREAM write queue is full due to internal flow control conditions), **write** will block until data can be accepted.
- If **O_NONBLOCK** is set and the STREAM cannot accept data, **write** will return -1 and set **errno** to **EAGAIN**.
- If **O_NONBLOCK** is set and part of the buffer has been written while a condition in which the STREAM cannot accept additional data occurs, **write** will terminate and return the number of bytes written.

In addition, **write** and **writew** 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 **writew** but reflects the prior error.

The **writew** 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 **writew** function will always write a complete area before proceeding to the next.

If **files** refers to a regular file and all of the **iov_len** members in the array pointed to by **iov** are 0, **writew** will return 0 and have no other effect. For other file types, the behaviour is unspecified.

If the sum of the **iov_len** values is greater than `SSIZE_MAX`, the operation fails and no data is transferred.

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.

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.

```
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.

Parameters

<i>Buffer</i>	Identifies the buffer containing the data to be written.
<i>Extension</i>	Provides communication with character device drivers that require additional information or return additional status. Each driver interprets the <i>Extension</i> 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 <i>Extension</i> parameter value is 0.
<i>FileDescriptor</i>	Identifies the object to which the data is to be written.
<i>iov</i>	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: <pre>caddr_t iov_base; size_t iov_len;</pre>
<i>iovCount</i>	Specifies the number of iovec structures pointed to by the <i>iov</i> 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.
EBADF	The <i>FileDescriptor</i> 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.
EFAULT	The <i>Buffer</i> parameter or part of the <i>iov</i> parameter points to a location outside of the allocated address space of the process.
EFBIG	(AIX 4.2 and later) 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 <i>FileDescriptor</i> parameter was negative; the <i>iovCount</i> parameter value was not between 1 and 16, inclusive; or one of the iov_len values in the iov array was negative or the sum overflowed a 32-bit integer.
EINVAL	The STREAM or multiplexer referenced by <i>FileDescriptor</i> is linked (directly or indirectly) downstream from a multiplexer.
EINTR	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.
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 <i>FileDescriptor</i> .

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 <i>NBytes</i> greater than zero and the starting offset is greater than or equal to the offset maximum established in the open file description associated with <i>FileDescriptor</i> .
EINVAL	The offset argument is invalid. The value is negative.
ESPIPE	files is associated with a pipe or FIFO.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **fcntl**, **dup**, or **dup2** subroutine, **fsync** subroutine, **ioctl** subroutine, **lockfx** subroutine, **lseek** subroutine, **open**, **openx**, or **creat** subroutine, **pathconf** subroutine, **pipe** subroutine, **poll** subroutine, **select** (“select Subroutine” on page 87) subroutine, **ulimit** (“ulimit Subroutine” on page 362) subroutine.

The **limits.h** file, **unistd.h** file.

The Input and Output Handling Programmer’s Overview in *AIX 5L Version 5.1 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 * wstrrchr (XString, Number)  
wchar_t *XString;  
int Number;
```

```
wchar_t * wstrpbrk (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 * wstrdup (XString1)
wchar_t *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 pointed 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 pointed to by the *XString1* parameter. Copying stops before *Number* **wchar_t** character if a null character is encountered in the string pointed 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 **wstrncmp** 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 **wstrcpy** 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 **wstrcpy** 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 **wstrrchr** 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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **malloc** subroutine, **strcat**, **strncat**, **strxfrm**, **strcpy**, **strncpy**, or **strdup** (“strcat, strncat, strxfrm, strcpy, strncpy, or strdup Subroutine” on page 255) subroutine, **strcmp**, **strncmp**, **strcasecmp**, **strncasecmp**, or **strcoll** (“strcmp, strncmp, strcasecmp, strncasecmp, or strcoll Subroutine” on page 257) subroutine, **strlen**, **strchr**, **strrchr**, **strpbrk**, **strspn**, **strcspn**, **strstr**, or **strtok** (“strlen, strchr, strrchr, strpbrk, strspn, strcspn, strstr, or strtok Subroutine” on page 265) subroutine.

List of String Manipulation Services in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

National Language Support Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Subroutines Overview in *AIX 5L Version 5.1 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**.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **atof**, **atoff**, **strtod**, **strtof** subroutine, **scanf**, **fscanf**, **sscanf** (“scanf, fscanf, sscanf, or wsscanf Subroutine” on page 82) subroutine, **strtol**, **strtoul**, **atol**, **atoi** (“strtol, strtoul, strtoll, strtoull, atol, or atoi Subroutine” on page 269) subroutine, **wstrtol**, **watol**, **watoi** (“wstrtol, watol, or watoi Subroutine”) subroutine.

Subroutines Overview in *AIX 5L Version 5.1 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 ( String, Pointer, Base)
```

```
wchar_t *String, **Pointer;
```

```
int Base;
```

```
long watol (String)
```

```
wchar_t *String;
```

```
int watoi (String)
```

```
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** (*String*) subroutine functions like **wstrtol** (*String*, (**wchar_t ****) null, **10**).

The **watoi** (*String*) subroutine functions like **(int) wstrtol** (*String*, (**wchar_t ****) null, **10**).

Note: Even if overflow occurs, it is ignored.

Parameters

<i>String</i>	Specifies the address of the string to scan.
<i>Pointer</i>	Specifies the address at which the pointer to the terminating character is stored.
<i>Base</i>	Specifies an integer value used as the base during conversion.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **atof**, **atoff**, **strtod**, **strtof** subroutine, **scanf**, **fscanf**, **sscanf** (“scanf, fscanf, sscanf, or wsscanf Subroutine” on page 82) subroutine, **strtol**, **strtoul**, **atol**, **atoi** (“strtol, strtoul, strtoll, strtoull, atol, or atoi Subroutine” on page 269) subroutine, **wstrtod**, **watof** (“wstrtod or watof Subroutine” on page 457) subroutine.

Subroutines Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **getpriority**, **setpriority**, or **nice** subroutine, **setpri** (“setpri Subroutine” on page 118) subroutine.

Chapter 2. Curses Subroutines

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 chtype ch);
int mvaddch(int y,
int x,
const chtype ch);
int mvwaddch(WINDOW *in,
const chtype ch);
int waddch(WINDOW *win,
const chtype 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 \hat{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** (“inch, mvinch, mvwinch, or winch Subroutine” on page 514) 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:

```
chtype x;  
x='H';  
addch(x);
```
2. To add the *x* character to *stdscr* at the coordinates *y* = 10, *x* = 5, enter:

```
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:

```
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:

```
WINDOW *my_window;  
waddch(my_window, 'x');
```
5. To add the character *x* in standout mode, enter:

```
waddch(my_window, 'x' | A_STANDOUT);
```

This allows 'x' to be highlighted, but leaves the rest of the window alone.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **inch**, **winch**, **mvinch**, or **mvwinch** (“inch, mvinch, mvwinch, or winch Subroutine” on page 514) subroutines, **wclrtoeol** (“clrtoeol or wclrtoeol Subroutine” on page 480) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Characters with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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

```
#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 n);
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

<i>Column</i>	Specifies the horizontal position to move the cursor to before adding the string.
<i>Line</i>	Specifies the vertical position to move the cursor to before adding the string.
<i>String</i>	Specifies the string to add.
<i>Window</i>	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:

```

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:

```

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:

```

mvwaddstr(my_window, 10, 5, "xyz");

```

4. To add the *xyz* string to the user-defined string at the current cursor location, enter:

```

waddstr(my_window, "xyz");

```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **addch** (“addch, mvaddch, mvwaddch, or waddch Subroutine” on page 461) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Characters with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

attroff, attron, attrset, wattroff, wattron, or wattrset Subroutine

Purpose

Restricted window attribute control functions.

Library

Curses Library (**libcurses.a**)

Syntax

```
#include <curses.h>
```

```
int attroff (int *attrs);
```

```
int attron (int *attrs);
```

```
int attrset (int *attrs);
```

```
int wattroff (WINDOW *win, int *attrs);
```

```
int wattron (WINDOW *win, int *attrs);
```

```
int wattrset (WINDOW *win, int *attrs);
```

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 window without affecting any others.

The **attron** and **wattron** subroutines turn on *attrs* in the current or specified window without affecting any others.

The **attrset** and **wattrset** subroutines set the background attributes of the current or specified window to *attrs*.

It is 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:

```
attroff(A_UNDERLINE);
```
2. To turn off the underlining attribute in the user-defined window `my_window`, enter:

```
wattroff(my_window, A_UNDERLINE);
```

For the **attron** or **wattron** subroutines:

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);
```

For the **attrset** or **wattrset** subroutines:

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);
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **standend** (“standend, standout, wstandend, or wstandout Subroutine” on page 579) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Setting Video Attributes and Curses Options in *AIX 5L Version 5.1 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)  
char *Attributes;
```

```
wattron( Window, Attributes)  
WINDOW *Window;  
char *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

<i>Attributes</i>	Specifies which attributes to turn on.
<i>Window</i>	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);
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Setting Video Attributes and Curses Options in *AIX 5L Version 5.1 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)
char *Attributes;
wattrset( Window, Attributes)
WINDOW *Window;
char *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

<i>Attributes</i>	Specifies which attributes to set.
<i>Window</i>	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);`

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Setting Video Attributes and Curses Options in *AIX 5L Version 5.1 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();
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **tcgetattr** (“tcgetattr Subroutine” on page 316) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Obsolete Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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();
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **flash** (“flash Subroutine” on page 497) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Setting Video Attributes and Curses Options in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

box Subroutine

Purpose

Draws borders from single-byte characters and renditions.

Library

Curses Library (**libcurses.a**)

Syntax

```
#include <curses.h>

int box(WINDOW *win,
        chtype verch,
        chtype 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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming, List of Curses Subroutines, and Windows in the Curses Environment in *AIX 5L Version 5.1 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

```
#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

color
**red*
**green*
**blue*
pair
f
b
value

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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **attroff** (“attroff, attron, attrset, wattroff, wattron, or wattrset Subroutine” on page 464) subroutine.

Curses Overview for Programming and Manipulating Video Attributes in *AIX 5L Version 5.1 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();`

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **getch** (“getch, mvgetch, mvwgetch, or wgetch Subroutine” on page 501) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Understanding Terminals with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

clear, erase, wclear or werase Subroutine

Purpose

Clears a window.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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:

```
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);`

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **douupdate** (“douupdate, refresh, wnoutrefresh, or wrefresh Subroutines” on page 602) subroutine, **erase** (“erase or werase Subroutine” on page 494) and **werase** (“erase or werase Subroutine” on page 494) subroutines, **clearok** (“clearok, idlok, leaveok, scrolllok, setscreg or wsetscreg Subroutine”) subroutine, **refresh** (“refresh or wrefresh Subroutine” on page 550) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Characters with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

clearok, idlok, leaveok, scrolllok, setscreg or wsetscreg 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 scrolllok(WINDOW *win,  
bool bf);
```

```
int setscreg(int top,  
int bot);
```

```
int wsetscreg(WINDOW *win,  
int top,  
int bot);
```

Description

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 **setscrreg** and **wsetscrreg** 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:
`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:

```
WINDOW *my_window;
wsetscrreg(my_window, 9, 29);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **douupdate** (“douupdate, refresh, wnoutrefresh, or wrefresh Subroutines” on page 602) subroutine, **scrll** (“scrll, scroll, wscrll Subroutine” on page 562) subroutine.

The **clear** subroutine, **refresh** or **wrefresh** (“refresh or wrefresh Subroutine” on page 550) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines and Manipulating Characters with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

clrrobot or wclrrobot Subroutine

Purpose

Erases the current line from the logical cursor position to the end of the window.

Library

Curses Library (**libcurses.a**)

Syntax

```
#include <curses.h>
int clrrobot(void);

int wclrrobot(WINDOW *win);
```

Description

The **clrrobot** and **wclrrobot** 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:

```
clrrobot();
```
2. To erase the lines below and to the right of the logical cursor in the user-defined window `my_window`, enter:

```
WINDOW *my_window;
wclrrobot(my_window);
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **douupdate** (“douupdate, refresh, wnoutrefresh, or wrefresh Subroutines” on page 602) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Characters with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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);
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **douupdate** (“douupdate, refresh, wnoutrefresh, or wrefresh Subroutines” on page 602) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Characters with Curses in *AIX 5L Version 5.1 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;
short *R, *G, *B;
```

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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **start_color** (“start_color Subroutine” on page 580) subroutine.

Curses Overview for Programming, Manipulating Video Attributes,

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

copywin Subroutine

Purpose

Copies a region of a window.

Library

Curses Library (**libcurses.a**)

Syntax

```
#include <curses.h>
```

```
int copywin(const WINDOW *srcwin,  
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

<i>*srcwin</i>	Points to the source window containing the region to copy.
<i>*dstwin</i>	Points to the destination window to copy into.
<i>sminrow</i>	Specifies the upper left row coordinate of the source region.
<i>smincol</i>	Specifies the upper left column coordinate of the source region.
<i>dminrow</i>	Specifies the upper left row coordinate of the destination region.

<i>dmincol</i>	Specifies the upper left column coordinate for the destination region.
<i>dmaxrow</i>	Specifies the lower right row coordinate for the destination region.
<i>dmaxcol</i>	Specifies the lower right column coordinate for the destination region.
<i>overlay</i>	Sets the type of copy. If set to TRUE the copy is nondestructive. 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.

Implementation Specifics

This routine is part of Base Operating System (BOS) Runtime.

Related Information

The **newpad** (“newpad, pnoutrefresh, prefresh, or subpad Subroutine” on page 533) and **overlay or overwrite** (“overlay or overwrite Subroutine” on page 543) subroutines.

Curses Overview for Programming, Manipulating Window Data with Curses Manipulating Characters with Curses, List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*

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:

Value of visibility	Appearance of Cursor
---------------------	----------------------

- 0 invisible
- 1 terminal-specific normal mode
- 2 terminal-specific high visibility mode

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:

```
cur_set(0);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*

Setting Video Attributes in *AIX 5L Version 5.1 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **doupage** ("doupdate, refresh, wnoutrefresh, or wrefresh Subroutines" on page 602), **endwin** ("endwin Subroutine" on page 494), **initscr** ("initscr and newterm Subroutine" on page 518), and the **setupterm** ("setupterm Subroutine" on page 568) subroutines.

Curses Overview for Programming, List of Curses Subroutines, Understanding Terminals with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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:

```
def_shell_mode();
```

This routine saves the "out of curses" state.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **setupterm** ("setupterm Subroutine" on page 568) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Understanding Terminals with Curses in *AIX 5L Version 5.1 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

```
#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 `fildev` parameter to a file descriptor, open for output, to the terminal device, before calling the **setupterm** subroutine. If the `erret` 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:

```
setupterm(char *)0, 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 `fildev` parameter, but does not destroy other information created by the **initscr**, **newterm**, or **setupterm** subroutines.

Parameters

**oterm*
**term*
fildev
**erret*
**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:

```
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:

```
restartterm("aixterm", 1, (int*)0);
```

For the **set_curterm** subroutine:

To set the **cur_term** variable to point to the **my_term** terminal, use:

```
TERMINAL *newterm; set_curterm(newterm);
```

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:

```
setupterm((char*) 0, 1, (int*) 0);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **baudrate** (“baudrate Subroutine” on page 468) subroutine, **longname** (“longname Subroutine” on page 528) subroutine, **putc** subroutine, **tgetent** (“tgetent, tgetflag, tgetnum, tgetstr, or tgoto Subroutine” on page 584) subroutine, **tigetflag** (“tigetflag, tigetnum, tigetstr, or tparm Subroutine” on page 588) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Understanding Terminals with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

delay_output Subroutine

Purpose

Sets 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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming, List of Curses Subroutines, Understanding Terminals with Curses in *AIX 5L Version 5.1 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:
`wdelch(my_window, 20, 30);`

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming, List of Curses Subroutines, Manipulating Characters with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

deleteln or wdeleteln Subroutine

Purpose

Deletes lines in a window.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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:

```
WINDOW *my_window;  
wdeleteLn(my_window);
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Characters with Curses in *AIX 5L Version 5.1 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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **derwin** (“derwin, newwin, or subwin Subroutine” on page 537) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Window Data with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

echo or noecho Subroutine

Purpose

Enables/disables terminal echo.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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:
echo();
2. To turn echoing off, use:
noecho();

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **wgetch** (“*getch, mvgetch, mvwgetch, or wgetch Subroutine*” on page 501) subroutine

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines and Understanding Terminals with Curses in *AIX 5L Version 5.1 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

```
#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:

```
echochar('I');
```

Implementation Specifics

This routine is part of Base Operating System (BOS) Runtime.

Related Information

The **addch**, **doupdate**, **echo_wchar**, **waddch**, **wmvaddch**, and **mvaddch** (“*addch, mvaddch, mvwaddch, or waddch Subroutine*” on page 461) subroutines.

Curses Overview for Programming and List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Characters with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

endwin Subroutine

Purpose

Suspends curses session.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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:

```
endwin();
```

Implementation Specifics

The **endwin** subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **douupdate** (“douupdate, refresh, wnoutrefresh, or wrefresh Subroutines” on page 602) subroutine, **initscr** (“initscr and newterm Subroutine” on page 518) subroutine, and **isendwin** subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Starting and Stopping Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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 *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);
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Characters with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

eraseschar, eraseswchar, killchar, and killwchar Subroutine

Purpose

Terminal environment query functions.

Library

Curses Library (**libcurses.a**)

Syntax

```
#include <curses.h>
char eraseschar(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();
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **clearok** ("clearok, idlok, leaveok, scrollok, setscreg or wsetscreg Subroutine" on page 476) subroutine, **tcgetattr** ("tcgetattr Subroutine" on page 316) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Obsolete Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs* describes how to query baud rate, set user-defined characters, and flush type-ahead characters.

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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **initscr** (“initscr and newterm Subroutine” on page 518) subroutine, **newterm** (“newterm Subroutine” on page 536) subroutine.

Curses Overview for Programming and List of Curses Subroutines in *AIX 5L Version 5.1 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();
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **beep** (“beep Subroutine” on page 468) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Setting Video Attributes and Curses Options in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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();
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Obsolete Curses Subroutines in *AIX 5L Version 5.1 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

<i>Window</i>	Points to a window.
<i>BegLine</i>	Identifies the beginning line in a range of lines to discard.
<i>NumLines</i>	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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Window Data with Curses in *AIX 5L Version 5.1 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Controlling the Cursor with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 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:

Mode	Action of getch Subroutines
NODELAY	Returns a value of ERR if there is no input waiting.
DELAY	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.
HALF-DELAY	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.

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(<i>n</i>)	Function key <i>F_n</i> , where <i>n</i> 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_CTAB	Clear 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_SFIND	shifted find key
KEY_SHELP	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

<i>Column</i>	Specifies the horizontal position to move the logical cursor to before getting the character.
<i>Line</i>	Specifies the vertical position to move the logical cursor to before getting the character.
<i>Window</i>	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:

```
mvwgetch();
```


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:

```
WINDOW *my_window;  
mwwgetch(my_window, 20, 30);
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **cbreak** (“cbreak, nocbreak, noraw, or raw Subroutine” on page 473), **douupdate** (“douupdate, refresh, wnoutrefresh, or wrefresh Subroutines” on page 602), and **insch** (“insch, mvinsch, mvwinsch, or winsch Subroutine” on page 519) subroutines, **keypad** (“keypad Subroutine” on page 523) subroutine, **meta** (“meta Subroutine” on page 529) subroutine, **nodelay** (“nodelay Subroutine” on page 540) subroutine, **echo** or **noecho** (“echo or noecho Subroutine” on page 492) subroutine, **notimeout** (“notimeout, timeout, wtimeout Subroutine” on page 541) subroutine, **ebreak** or **nocbreak** (“cbreak, nocbreak, noraw, or raw Subroutine” on page 473) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Characters with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

getmaxyx Subroutine

Purpose

Returns the size of a window.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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

<i>Window</i>	Identifies the window whose size to get.
<i>Y</i>	Contains the number of rows in the window.
<i>X</i>	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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Controlling the Cursor with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 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 *st,
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 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** subroutine, 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:

```
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:

```
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;
mwgetstr(my_window, 20, 30, my_string);
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **beep** (“beep Subroutine” on page 468) subroutine, **getch** (“getch, mvgetch, mvwgetch, or wgetch Subroutine” on page 501) subroutine, **keypad** (“keypad Subroutine” on page 523) subroutine, **nodelay** (“nodelay Subroutine” on page 540) subroutine, **wgetch** (“getch, mvgetch, mvwgetch, or wgetch Subroutine” on page 501) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Characters with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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 **ripline** 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.

Implementation Specifics

This routine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Controlling the Cursor with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 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

```
#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:

```
WINDOW *my_window;  
int line, column;  
getyx(my_window, line, column);
```

Implementation Specifics

This macro is part of Base Operating System (BOS) Runtime.

Related Information

Controlling the Cursor with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

halfdelay Subroutine

Purpose

Controls input character delay mode.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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** (“cbreak, nocbreak, noraw, or raw Subroutine” on page 473) 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();
```

Implementation Specifics

This routine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Video Attributes in *AIX 5L Version 5.1 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:

```
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:

```
int insert_line;
insert_line = has_il();
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 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

```
#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:

```
int insert_line;
insert_line = has_il();
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Understanding Terminals with Curses in *AIX 5L Version 5.1 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

```
#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:

```
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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Setting Video Attributes and Curses Options in *AIX 5L Version 5.1 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:

```
chtype 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);
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS).

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

init_color Subroutine

Purpose

Changes a color definition.

Library

Curses Library (**libcurses.a**)

Syntax

```
#include < curses.h >
```

```
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);`

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **start_color** (“start_color Subroutine” on page 580) subroutine.

Curses Overview for Programming and Manipulating Video Attributes in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

init_pair Subroutine

Purpose

Changes a color-pair definition.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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);
```

Implementation Specifics

This routine is part of Base Operating System (BOS) Runtime.

Related Information

The **init_color** (“init_color Subroutine” on page 516) subroutine, **start_color** (“start_color Subroutine” on page 580) subroutine.

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.

```
initscr          CURSES          Curses Interfaces
```

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:

```
initscr();
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **douupdate** (“douupdate, refresh, wnoutrefresh, or wrefresh Subroutines” on page 602) subroutine, **del_curterm** (“del_curterm, restartterm, set_curterm, or setupterm Subroutine” on page 486) subroutine, **filter** (“filter Subroutine” on page 496) subroutine, **slk_attroff** (“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” on page 570) subroutine, **setupterm** (“setupterm Subroutine” on page 568) subroutine.

Curses Overview for Programming, Initializing Curses, List of Curses Subroutines in *AIX 5L Version 5.1 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

```
#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:

```
chtype x;
insch(x);
```

2. To insert the character `x` into the user-defined window `my_window`, enter:

```
WINDOW *my_window
chtype x;
winsch(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:

```
chtype x;
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 `my_window`, enter:

```
WINDOW *my_window;
chtype x;
mwinsch(my_window, 10, 5, x);
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Characters with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

insertln or winsertln 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);
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Characters with Curses in *AIX 5L Version 5.1 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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Setting Video Attributes and Curses Options in *AIX 5L Version 5.1 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:

Input	Format of Returned String
Visible character	The same character
Control character	^X

Meta-character (keyname only)	M-X
Key value defined in <curses.h> (keyname only)	KEY_name
None of the above	UNKNOWN KEY

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

```
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**.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **meta** (“meta Subroutine” on page 529) and **wgetch** (“getch, mvgetch, mvwgetch, or wgetch Subroutine” on page 501) subroutines.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

keypad Subroutine

Purpose

Enables or disables abbreviation of function keys.

Library

Curses Library (**libcurses.a**)

Syntax

```
#include <curses.h>
```

```
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:

```
WINDOW *my_window;
keypad(my_window, TRUE);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **getch** (“getch, mvgetch, mvwgetch, or wgetch Subroutine” on page 501) subroutine.

The **terminfo** file format.

Curses Overview for Programming, List of Curses Subroutines, Setting Video Attributes and Curses Options in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

killchar or killwchar Subroutine

Purpose

Terminal environment query functions.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Portability with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs* describes how to query baud rate, set user-defined characters, and flush type-ahead characters.

_lazySetErrorHandler Subroutine

Purpose

Installs an error handler into the lazy loading runtime system for the current process.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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:

```
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:

```
WINDOW *my_window;  
mvwgetch(my_window, 20, 30);
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **keypad** (“keypad Subroutine” on page 523) subroutine, **meta** (“meta Subroutine” on page 529) subroutine, **nodelay** (“nodelay Subroutine” on page 540) subroutine, **echo** or **noecho** (“echo or noecho Subroutine” on page 492) subroutine, **notimeout** (“notimeout, timeout, wtimeout Subroutine” on page 541) subroutine, **ebreak** or **nocbreak** (“cbreak, nocbreak, noraw, or raw Subroutine” on page 473) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Characters with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 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

```
#include <curses.h>
```

```
leaveok( Window, Flag)  
WINDOW *Window;  
bool Flag;
```

Description

The **leaveok** subroutine controls cursor placement after a call to the **refresh** (“refresh or wrefresh Subroutine” on page 550) 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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **refresh** (“refresh or wrefresh Subroutine” on page 550) subroutine.

Controlling the Cursor with Curses, Curses Overview for Programming, List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

longname Subroutine

Purpose

Returns the verbose name of a terminal.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **initscr** (“initscr and newterm Subroutine” on page 518) subroutine, **newterm** (“newterm Subroutine” on page 536) subroutine, **setupterm** (“setupterm Subroutine” on page 568) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Understanding Terminals with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

makenew Subroutine

Purpose

Creates a new window buffer and returns a pointer.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Understanding Windows with Curses in *AIX 5L Version 5.1 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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **getch** (“getch, mvgetch, mvwgetch, or wgetch Subroutine” on page 501) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Characters with Curses in *AIX 5L Version 5.1 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 550) 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);
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **getch** (“getch, mvgetch, mvwgetch, or wgetch Subroutine” on page 501) and **refresh** (“refresh or wrefresh Subroutine” on page 550) subroutines.

Controlling the Cursor with Curses, Curses Overview for Programming, List of Curses Subroutines in *AIX 5L Version 5.1 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
oldcol
oldrow

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*.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **douupdate** ("douupdate, refresh, wnoutrefresh, or wrefresh Subroutines" on page 602) subroutine, **is_linetouched** ("is_linetouched, is_wintouched, touchline, touchwin, untouchwin, or wtouchin Subroutine" on page 592) subroutine, **move** ("move or wmove Subroutine" on page 530) subroutine, **refresh** ("refresh or wrefresh Subroutine" on page 550) subroutine.

Controlling the Cursor with Curses, Curses Overview for Programming, List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **derwin** (“`derwin`, `newwin`, or `subwin` Subroutine” on page 537) subroutine, **douupdate** (“`douupdate`, `refresh`, `wnoutrefresh`, or `wrefresh` Subroutines” on page 602) subroutine, **is_linetouched** (“`is_linetouched`, `is_wintouched`, `touchline`, `touchwin`, `untouchwin`, or `wtouchin` Subroutine” on page 592) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Window Data with Curses in *AIX 5L Version 5.1 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 545) or **pnoutrefresh** (“prefresh or pnoutrefresh Subroutine” on page 545) 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 *pminrow* and *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 *pminrow*, *pmincol*, *sminrow* or *smincol* are treated as if they were zero.

Parameters

```

ncols
nlines
begin_x
begin_y
*orig
*pad
pminrow
pmincol
sminrow
smincol
smaxrow
smaxcol

```

Return Values

Upon successful completion, the **newpad** and **subpad** subroutines return a pointer to the pad structure. Otherwise, they return a null pointer.

Upon successful completion, the **pnoutrefresh** and **prefresh** subroutines return OK. Otherwise, they return ERR.

Examples

For the **newpad** subroutine:

1. To create a new pad and save the pointer to it in `my_pad`, enter:

```
WINDOW *my_pad;
my_pad = newpad(5, 10);
```

`my_pad` is now a pad 5 lines deep, 10 columns wide.

2. To create a pad and save the pointer to it in `my_pad`, which is flush with the right side of the terminal, enter:

```
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.

3. To create a pad and save the pointer to it in `my_pad`, which fills the entire terminal, enter:

```
WINDOW *my_pad;
my_pad = newpad(0, 0);
```

`my_pad` is now a pad that fills the entire terminal.

4. To create a very large pad and display part of it on the screen, enter;

```
WINDOW *my_pad;
my_pad = newpad(120,120);
prefresh (my_pad, 0,0,0,0,20,30);
```

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

```
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();
```

For the **subpad** subroutine:

To create a subpad, use:

```
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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **derwin** (“derwin, newwin, or subwin Subroutine” on page 537) subroutine, **douupdate** (“douupdate, refresh, wnoutrefresh, or wrefresh Subroutines” on page 602) subroutine, **is_linetouched** (“is_linetouched, is_wintouched, touchline, touchwin, untouchwin, or wtouchin Subroutine” on page 592) subroutine, **prefresh** (“prefresh or pnoutrefresh Subroutine” on page 545) or **pnoutrefresh** (“prefresh or pnoutrefresh Subroutine” on page 545) subroutine, and **subpad** (“subpad Subroutine” on page 581) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Windows in the Curses Environment in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

newterm Subroutine

Purpose

Initializes curses and its data structures for a specified terminal.

Library

Curses Library (**libcurses.a**)

Syntax

```
#include <curses.h>

SCREEN *newterm(
    Type,
    OutFile, InFile)
char *Type;
FILE *OutFile, *InFile;
```

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:

```
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:

```
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:

```
fdifile = fopen("/dev/lft0", "r");  
fdofile = fopen("/dev/tty0", "w");
```

```
SCREEN *my_terminal2;  
my_terminal2 = newterm("ibm3151",fdofile, fdifile);
```

3. To use `stdin` for input and `stdout` for output, do the following:

```
char termname [] = "terminaltype";  
SCREEN *my_terminal;  
my_terminal = newterm(termname,stdout,stdin);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **endwin** (“endwin Subroutine” on page 494) subroutine, **initscr** (“initscr and newterm Subroutine” on page 518) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Initializing Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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 origin.) 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:

```
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:

```
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:

```
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:

```
WINDOW *my_window, *my_sub_window;
my_window = newwin ("derwin, newwin, or subwin Subroutine" on page 537)
(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

```
WINDOW *my_window, *my_sub_window;
my_window =
newwin ("derwin, newwin, or subwin Subroutine" on page 537)(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 537)
(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`.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **endwin** ("endwin Subroutine" on page 494), **initscr** ("initscr and newterm Subroutine" on page 518) subroutines.

Curses Overview for Programming, List of Curses Subroutines, Windows in the Curses Environment in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

nl or nonl Subroutine

Purpose

Enables/disables newline translation.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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 **nonnl** 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();`

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **refresh** (“refresh or wrefresh Subroutine” on page 550) subroutine, **waddch** (“addch, mvaddch, mvwaddch, or waddch Subroutine” on page 461) subroutine.

Curses Overview for Programming, Understanding Terminals with Curses, List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **halfdelay** (“halfdelay Subroutine” on page 510) subroutine, **wgetch** (“getch, mvgetch, mvwgetch, or wgetch Subroutine” on page 501) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Characters with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*

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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **getch** (“getch, mvgetch, mvwgetch, or wgetch Subroutine” on page 501), **halfdelay** (“halfdelay Subroutine” on page 510), **nodelay** (“nodelay Subroutine” on page 540), and **notimeout** (“notimeout, timeout, wtimeout Subroutine” on page 541) subroutines.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Characters with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Getting Characters in *AIX 5L Version 5.1 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

```
WINDOW *dstwin);  
int overwrite(const WINDOW *srcwin,  
WINDOW *dstwin);
```

Description

The **overlay** and **overwrite** subroutines overlay *srcwin* on top of *dstwin*. The *srcwin* 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:

```
WINDOW *my_window, *other_window;  
overlay(my_window, other_window);
```

2. To copy *my_window* on top of *other_window*, including spaces, use:

```
WINDOW *my_window, *other_window;  
overwrite(my_window, other_window);
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **copywin** ("copywin Subroutine" on page 482) subroutine.

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*.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **start_color** (“start_color Subroutine” on page 580) subroutine, **init_pair** (“init_pair Subroutine” on page 517) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Video Attributes, Working with Color in *AIX 5L Version 5.1 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

```
#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 550) and **wnoutrefresh** (“doupdate, refresh, wnoutrefresh, or wrefresh Subroutines” on page 602) 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();
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Window Data with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

printw, wprintw, mvprintw, or mvwprintw Subroutine

Purpose

Performs a **printw** command on a window using the specified format control string.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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;
mvprintw(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 461) 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

<i>Argument</i>	Specifies the item to print. See the printf subroutine for more details.
<i>Column</i>	Specifies the horizontal position to move the cursor to before printing.
<i>Format</i>	Specifies the format for printing the <i>Argument</i> parameter. See the printf subroutine.
<i>Line</i>	Specifies the vertical position to move the cursor to before printing.
<i>Window</i>	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:

```

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:

```

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:

```

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:

```

int x, y;
WINDOW *my_window;
mvwprintw(my_window, 5, 10, "%d%d", x, y);

```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **waddch** (“addch, mvaddch, mvwaddch, or waddch Subroutine” on page 461) subroutine, **printf** subroutine.

The **printf** command.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Characters with Curses in *AIX 5L Version 5.1 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 affcnt,
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 files 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 **tparam** 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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **douupdate** (“douupdate, refresh, wnoutrefresh, or wrefresh Subroutines” on page 602) subroutine, **is_linetouched** (“is_linetouched, is_wintouched, touchline, touchwin, untouchwin, or wtouchin Subroutine” on page 592) subroutine, **putchar** subroutine, **tgetent** (“tgetent, tgetflag, tgetnum, tgetstr, or tgoto Subroutine” on page 584) subroutine, **tigetflag** (“tigetflag, tigetnum, tigetstr, or tparm Subroutine” on page 588) subroutine, **tputs** (“putp, tputs Subroutine” on page 548) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Understanding Terminals with Curses in *AIX 5L Version 5.1 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 473

page 473) 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();
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **getch** (“getch, mvgetch, mvwgetch, or wgetch Subroutine” on page 501) subroutine, **cbreak** or **nocbreak** (“cbreak, nocbreak, noraw, or raw Subroutine” on page 473) subroutine

Curses Overview for Programming, List of Curses Subroutines, Understanding Terminals with Curses in *AIX 5L Version 5.1 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** ("leaveok Subroutine" on page 526) subroutine is enabled.

The **refresh** and **wrefresh** subroutines call two other subroutines to perform the refresh operation. First, the **wnoutrefresh** ("doupdate, refresh, wnoutrefresh, or wrefresh Subroutines" on page 602) subroutine copies the designated window structure to the terminal. Then, the **doupdate** ("doupdate, refresh, wnoutrefresh, or wrefresh Subroutines" on page 602) 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:

```
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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **doupdate** ("doupdate, refresh, wnoutrefresh, or wrefresh Subroutines" on page 602) subroutine, **leaveok** ("leaveok Subroutine" on page 526) subroutine, **wnoutrefresh** ("leaveok Subroutine" on page 526) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Characters with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

reset_prog_mode Subroutine

Purpose

Restores the terminal to program mode.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Understanding Terminals with Curses in *AIX 5L Version 5.1 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **endwin** ("endwin Subroutine" on page 494) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Understanding Terminals with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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** (“endwin Subroutine” on page 494) subroutine, and should normally not be called directly by a program.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Understanding Terminals with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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();
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **def_prog_mode** (“def_prog_mode, def_shell_mode, reset_prog_mode or reset_shell_mode Subroutine” on page 484) subroutine, **endwin** (“endwin Subroutine” on page 494) subroutine, **savetty** (“savetty Subroutine” on page 556) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Understanding Terminals with Curses in *AIX 5L Version 5.1 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

<i>Term</i>	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.
<i>FileNumber</i>	Specifies the output file's file descriptor (1 equals standard out).
<i>ErrorCode</i>	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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Example

To restart an **aixterm** after a previous memory save and exit on error with a message, enter:

```
restartterm("aixterm", 1, (int*)0);
```

Implementation Specifics

This routine is part of Base Operating System (BOS) Runtime

Prerequisite Information

Curses Overview for Programming and Understanding Terminals with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs* .

Related Information

The **setupterm** ("setupterm Subroutine" on page 568) subroutine.

ripcoffline Subroutine

Purpose

Reserves a line for a dedicated purpose.

Library

Curses Library (**libcurses.a**)

Syntax

```
#include  
<curses.h>  
  
int  
ripcoffline(int line,  
int (*init)(WINDOW *win,  
int columns));
```

Description

The **ripcoffline** subroutine reserves a screen line for use by the application.

Any call to the **ripcoffline** subroutine must precede the call to the **initscr** or **newterm** subroutine. If line is positive, one line is removed from the beginning of stdscr; 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:

```
#include <curses.h>
ripoffline(1,initfunc);
ripoffline(1,initfunc);
ripoffline(1,initfunc);

initscr();
```

Implementation Specifics

This routine is part of Base Operating System (BOS) Runtime.

Related Information

The `doupdate` (“`doupdate`, `refresh`, `wnoutrefresh`, or `wrefresh` Subroutines” on page 602) 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” on page 570) subroutine, `initscr` (“`initscr` and `newterm` Subroutine” on page 518) subroutine, `newterm` (“`newterm` Subroutine” on page 536) subroutine.

Curses Overview for Programming and List of Curses Subroutines in *AIX 5L Version 5.1 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **initscr** (“initscr and newterm Subroutine” on page 518) subroutine, **resetty** (“resetty, savetty Subroutine” on page 553) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Understanding Terminals with Curses in *AIX 5L Version 5.1 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

<i>Argument</i>	Specifies the input to read.
<i>Column</i>	Specifies the vertical coordinate to move the logical cursor to before performing the scan.
<i>Format</i>	Specifies the conversion specifications to use to interpret the input. For more information about this parameter, see the discussion of the <i>Format</i> parameter in the scanf (“scanf, fscanf, sscanf, or wsscanf Subroutine” on page 82) subroutine.
<i>Line</i>	Specifies the horizontal coordinate to move the logical cursor to before performing the scan.
<i>Window</i>	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.

```
int id;
char deptname[25];

mvprintw(5,0,"Enter your i.d. followed by the department name:\n");
refresh();
scanw("%d %s", &id, deptname);
mvprintw(7,0,"i.d.: %d, Name: %s\n", id, deptname);
refresh();
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **wgetstr** (“getnstr, getstr, mvgetnstr, mvgetstr, mvwgetnstr, mvwgetstr, wgetnstr, or wgetstr Subroutine” on page 506) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Characters with Curses in *AIX 5L Version 5.1 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

```
#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:

```
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:

```
scr_restore("/tmp/virtual.dump");  
doupdate();
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **doupdate** (“`doupdate`, `refresh`, `wnoutrefresh`, or `wrefresh` Subroutines” on page 602) subroutine, **endwin** (“`endwin` Subroutine” on page 494) subroutine, **open** subroutine, **read** (“`read`, `readx`, `readv`, `readvx`, or `pread` Subroutine” on page 9) subroutine, **write** (“`write`, `writex`, `writev`, `writevx` or `pwrite` Subroutines” on page 449) subroutine, **scr_init** (“`scr_init` Subroutine” on page 560) subroutine, **scr_restore** (“`scr_restore` Subroutine” on page 561) subroutine.

Curses Overview for Programming, Manipulating Window Data with Curses, Understanding Terminals with Curses and List of Curses Subroutines in *AIX 5L Version 5.1 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

```
#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**, **slk_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** (“scr_dump, scr_init, scr_restore, scr_set Subroutine” on page 558) subroutine, **scr_restore** (“scr_restore Subroutine”) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Window Data with Curses in *AIX 5L Version 5.1 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();
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **scr_dump** (“scr_dump, scr_init, scr_restore, scr_set Subroutine” on page 558) subroutine, **scr_init** (“scr_init Subroutine” on page 560) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Understanding Terminals with Curses, Manipulating Video Attributes in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

scr1, scroll, wscr1 Subroutine

Purpose

Scrolls a Curses window.

Library

Curses Library (**libcurses.a**)

Syntax

```
#include <curses.h>
```

```
int scr1  
(int n);
```

```
int scroll  
(WINDOW *win);
```

```
int wscr1  
(WINDOW *win,  
int n);
```

Description

The **scroll** subroutine scrolls win one line in the direction of the first line

The **scr1** and **wscr1** 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 **setscreg** 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:

```
WINDOW *my_window;  
scroll(my_window);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **scrollok** (“scrollok Subroutine”) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Characters with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

scrollok Subroutine

Purpose

Enables or disables scrolling.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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** (“idlok Subroutine” on page 513) 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:

```
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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **idllok** (“idllok Subroutine” on page 513) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Characters with Curses in *AIX 5L Version 5.1 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

```
#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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **setupterm** (“setupterm Subroutine” on page 568) 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 **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

<i>Bmargin</i>	Specifies the last line number in the scrolling region.
<i>Tmargin</i>	Specifies the first line number in the scrolling region (0 is the top line of the window.)
<i>Window</i>	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:

```
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:

```
WINDOW *my_window;  
wsetscrreg(my_window, 9, 29);
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **idlok** (“idlok Subroutine” on page 513) subroutine, **scrollok** (“scrollok Subroutine” on page 563) subroutine, **wrefresh** (“refresh or wrefresh Subroutine” on page 550) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Characters with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

setsyx Subroutine

Purpose

Sets the coordinates of the virtual screen cursor.

Library

Curses Library (**libcurses.a**)

Syntax

```
#include <curses.h>
```

```
setsyx( Y, X)  
int Y, 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **douupdate** (“douupdate, refresh, wnoutrefresh, or wrefresh Subroutines” on page 602) subroutine, **getsyx** (“getsyx Subroutine” on page 508) subroutine, **leaveok** (“leaveok Subroutine” on page 526) subroutine, **wnoutrefresh** (“douupdate, refresh, wnoutrefresh, or wrefresh Subroutines” on page 602) subroutine.

Controlling the Cursor with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Curses Overview for Programming and List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

set_term Subroutine

Purpose

Switches between screens.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **initscr** (“initscr and newterm Subroutine” on page 518) subroutine, **newterm** (“newterm Subroutine” on page 536) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Understanding Terminals with Curses in *AIX 5L Version 5.1 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

<i>ErrorCode</i>	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.
<i>FileNumber</i>	Specifies the output files file descriptor (1 equals standard output).
<i>Term</i>	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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **termdef** ("termdef Subroutine" on page 322) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Understanding Terminals with Curses in *AIX 5L Version 5.1 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

<i>Column</i>	Specifies the horizontal coordinate of the terminal at which to dump the string.
<i>First</i>	Specifies the beginning string address of the string to dump to the terminal.
<i>Last</i>	Specifies the end string address of the string to dump to the terminal.
<i>Line</i>	Specifies the vertical coordinate of the terminal at which to dump the string.
<i>String</i>	Specifies the string to dump to the terminal.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming, List of Curses Subroutines, Manipulating Characters with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

slk_atroff, 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_atroff
(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 soft 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_sttr_set**, and **slk_attroff** subroutines correspond to the **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

attrs

**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:

```
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:

```
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:

```
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:

```
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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **attroff** (“attroff, attron, attrset, wattroff, watron, or wattreset Subroutine” on page 464) subroutine, **riproffline** (“riproffline Subroutine” on page 555) subroutine, **wcswidth** (“wcswidth Subroutine” on page 409) subroutine, **slk_init** (“slk_init Subroutine”) subroutine, **slk_set** (“slk_set Subroutine” on page 577) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Video Attributes in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

slk_init Subroutine

Purpose

Initializes soft function-key labels.

Library

Curses Library (**libcurses.a**)

Syntax

```
#include <curses.h>
```

```
slk_init( Labfmt)
int Labfmt;
```

Description

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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **initscr** (“initscr and newterm Subroutine” on page 518) subroutine, **newterm** (“newterm Subroutine” on page 536) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Soft Labels in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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( LabNum)  
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 **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:

```
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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **slk_init** (“slk_init Subroutine” on page 573) subroutine and **slk_set** (“slk_set Subroutine” on page 577) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Video Attributes in *AIX 5L Version 5.1 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

```
#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:

```
slk_set(8, "Insert", 1);
slk_noutrefresh();
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **slk_init** (“slk_init Subroutine” on page 573) subroutine, **slk_refresh** (“slk_refresh Subroutine”) subroutine, **wrefresh** (“refresh or wrefresh Subroutine” on page 550) subroutine.

Curses Overview for Programming, Manipulating Video Attributes, List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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);
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();
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **slk_init routine** (“slk_init Subroutine” on page 573) subroutine, **slk_set routine** (“slk_set Subroutine” on page 577) subroutine, **slk_noutrefresh** (“slk_noutrefresh Subroutine” on page 575) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Video Attributes in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **slk_init** (“slk_init Subroutine” on page 573) subroutine, **slk_clear** (“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” on page 570) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Soft Labels in *AIX 5L Version 5.1 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

<i>LabNum</i>	Specifies the label number. The value can range from 1 to 8.
<i>LabStr</i>	Specifies the string (name) to put on the label. If the string is NULL, the label is blank.
<i>LabFmt</i>	Specifies the label alignment. The following values are valid:
0	Left-justified
1	Centered
2	Right-justified

Example

```
slk_set(2, "Quit", 1);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **slk_init** (“slk_init Subroutine” on page 573) routine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Video Attributes in *AIX 5L Version 5.1 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **slk_init** (“slk_init Subroutine” on page 573) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Video Attributes in *AIX 5L Version 5.1 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

```
#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:

```
standout();
```

This example is functionally equivalent to:

```
attron(A_STANDOUT);
```

2. To turn on the **standout** attribute in the user-defined window `my_window`, enter:

```
WINDOW *my_window;  
wstandout(my_window);
```

This example is functionally equivalent to:

```
wattron(my_window, A_STANDOUT);
```

3. To turn off the **standout** attribute in the default window, enter:

```
standend();
```

This example is functionally equivalent to:

```
attroff(A_STANDOUT);
```

4. To turn off the **standout** attribute in the user-defined window `my_window`, enter:

```
WINDOW *my_window;  
wstandend(my_window);
```

This example is functionally equivalent to:

```
wattroff(my_window, A_STANDOUT);
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **attroff**, **attron**, or **wattroff** (“attroff, attron, attrset, wattroff, wattron, or wattrset Subroutine” on page 464) subroutines.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Video Attributes in *AIX 5L Version 5.1 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:

COLOR_BLACK	0
COLOR_BLUE	1
COLOR_GREEN	2
COLOR_CYAN	3
COLOR_RED	4
COLOR_MAGENTA	5
COLOR_YELLOW	6
COLOR_WHITE	7

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:

```
start_color();
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **has_colors** ("has_colors Subroutine" on page 511) subroutine, **can_change_color** ("can_change_color, color_content, has_colors,init_color, init_pair, start_color or pair_content Subroutine" on page 470) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Manipulating Video Attributes in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

subpad Subroutine

Purpose

Creates a subwindow within a pad.

Library

Curses Library (**libcurses.a**)

Syntax

```
#include <curses.h>
```

```
WINDOW *subpad(Orig, NLines, NCols, Begin_Y, Begin_X)
```

```
WINDOW * Orig;
```

```
int NCols, NLines, Begin_Y, Begin_X;
```

Description

The **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 **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 **touchwin** or **touchline** subroutine on the parent pad before refreshing the parent pad. Use the **prefresh** subroutine to refresh a pad.

Parameters

<i>Orig</i>	Points to the parent pad.
<i>NLines</i>	Specifies the number of lines (rows) in the subpad.
<i>NCols</i>	Specifies the number of columns in the subpad.
<i>Begin_Y</i>	Identifies the upper left-hand row coordinate of the subpad relative to the parent pad.
<i>Begin_X</i>	Identifies the upper left-hand column coordinate of the subpad relative to the parent pad.

Examples

To create a subpad, use:

```
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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming, List of Curses Subroutines, Windows in the Curses Environment in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

subwin Subroutine

Purpose

Creates a subwindow within an existing window.

Library

Curses Library (**libcurses.a**)

Syntax

```
#include <curses.h>
WINDOW *subwin (ParentWindow, NumLines, NumCols, Line, Column)
WINDOW * ParentWindow ;
int 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** ("touchwin Subroutine" on page 595) or **touchline** subroutine on the parent window before calling the **wrefresh** ("refresh or wrefresh Subroutine" on page 550) subroutine on the parent window.

Changes to one window will affect the character image of both windows.

Parameters

<i>NumCols</i>	Indicates the number of vertical columns in the subwindow's width. If 0 is passed as the <i>NumCols</i> value, the subwindow runs from the <i>Column</i> to the right edge of its parent window.
<i>NumLines</i>	Indicates the number of horizontal lines in the subwindow's height. If 0 is passed as the <i>NumLines</i> parameter, then the subwindow runs from the <i>Line</i> to the bottom of its parent window.
<i>ParentWindow</i>	Specifies the subwindow's parent.
<i>Column</i>	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.
<i>Line</i>	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).

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 537)
              (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 537)
              (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 537)  
            (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`.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **touchwin** ("touchwin Subroutine" on page 595), **newwin** ("derwin, newwin, or subwin Subroutine" on page 537), and **wrefresh** ("refresh or wrefresh Subroutine" on page 550) subroutines.

Curses Overview for Programming, List of Curses Subroutines, Windows in the Curses Environment in *AIX 5L Version 5.1 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **putc**, **setupterm** (“setupterm Subroutine” on page 568), **tigetflg** (“tigetflag, tigetnum, tigetstr, or tparm Subroutine” on page 588) subroutines.

Curses Overview for Programming, List of Curses Subroutines, Understanding Terminals with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

tigetflag Subroutine

Purpose

Library

Curses Library (**libcurses.a**)

Syntax

```
#include <curses.h>

bool tigetflag( ID)
char *ID;
```

Description

The **tigetflag** 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Understanding Terminals with Curses in *AIX 5L Version 5.1 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( ID)  
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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Understanding Terminals with Curses in *AIX 5L Version 5.1 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Understanding Terminals with Curses in *AIX 5L Version 5.1 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** (“tparm Subroutine” on page 596) 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.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **tparm** (“tparm Subroutine” on page 596) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Understanding Terminals with Curses in *AIX 5L Version 5.1 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

```
#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);
```

Implementation Specifics

This routine is part of Base Operating System (BOS) Runtime.

Related Information

The **def_prog_mode** ("def_prog_mode, def_shell_mode, reset_prog_mode or reset_shell_mode Subroutine" on page 484), **tgetent** ("tgetent, tgetflag, tgetnum, tgetstr, or tgoto Subroutine" on page 584), and **putp** ("putp, tputs Subroutine" on page 548) subroutines.

Curses Overview for Programming, List of Curses Subroutines

Understanding Terminals with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

tigetnum Subroutine

Purpose

Gets the value of terminal's numeric capability.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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:

```
rc = tigetnum("nlab");
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Understanding Terminals with Curses in *AIX 5L Version 5.1 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.

Implementation Specifics

This routine is part of Base Operating System (BOS) Runtime.

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 5.1 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

```
#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_wintouchwin** 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:

```
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:

```
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`.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **douupdate** (“`douupdate`, `refresh`, `wnoutrefresh`, or `wrefresh` Subroutines” on page 602) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Understanding Windows with Curses in *AIX 5L Version 5.1 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

```
#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

<i>Window1</i>	Specifies the first window as changed.
<i>Window2</i>	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);`

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Understanding Windows with Curses in *AIX 5L Version 5.1 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

```
#include <curses.h>
```

```
touchwin( Window)
```

```
WINDOW *Window;
```

Description

The **touchwin** ("touchwin Subroutine") 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** ("is_linetouched, is_wintouched, touchline, touchwin, untouchwin, or wtouchin Subroutine" on page 592) 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 redisplay 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:

```
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`.

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **touchline** (“`is_linetouched`, `is_wintouched`, `touchline`, `touchwin`, `untouchwin`, or `wtouchin` Subroutine” on page 592) subroutine, **wrefresh** (“`refresh` or `wrefresh` Subroutine” on page 550) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Windows in the Curses Environment in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

tparm Subroutine

Purpose

Applies parameters (padding) to a terminal capability.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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

<i>Parm#</i>	Specifies the parameters (up to nine) to instantiate.
<i>TermCap</i>	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** (“`tputs` Subroutine” on page 597) 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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Understanding Terminals with Curses in *AIX 5L Version 5.1 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

LinesAffected

Specifies the number of lines affected, or specifies 1 if not applicable.

PutcLikeSub

Specifies a **putchar**-like subroutine through which the characters are passed one at a time.

String

Specifies the string to which to add padding information.

Examples

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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **tparm** (“tparm Subroutine” on page 596) subroutine.

Curses Overview for Programming, List of Curses Subroutines, Understanding Terminals with Curses in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

typeahead Subroutine

Purpose

Controls checking for typeahead.

Library

Curses Library (**libcurses.a**)

Syntax

```
#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);
```

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **douupdate** (“douupdate, refresh, wnoutrefresh, or wrefresh Subroutines” on page 602), **getch** (“getch, mvgetch, mvwgetch, or wgetch Subroutine” on page 501), and **initscr** (“initscr and newterm Subroutine” on page 518) subroutines.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Setting Video Attributes and Curses Options in *AIX 5L Version 5.1 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  
(chtype 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 \backslash 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".

Implementation Specifics

This subroutine is part of Base Operating System (BOS) Runtime.

Related Information

The **keyname** ("keyname, key_name Subroutine" on page 522) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Characters with Curses in *AIX 5L Version 5.1 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);
```


Implementation Specifics

This subroutine is part of Base Operation System (BOS) Runtime.

Related Information

The **getch** or **wgetch** (“getch, mvgetch, mvwgetch, or wgetch Subroutine” on page 501) subroutine.

Curses Overview for Programming and List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Characters with Curses in *AIX 5L Version 5.1 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

```
#include <curses.h>

int vidattr
(chtype attr);

int vid_attr
(attr_t attr,
short color_pair_number,
void *opt);

int vidputs
(chtype 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);
```

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **douupdate** (“**douupdate**, **refresh**, **wnoutrefresh**, or **wrefresh** Subroutines”), **is_linetouched** (“**is_linetouched**, **is_wintouched**, **touchline**, **touchwin**, **untouchwin**, or **wtouchin** Subroutine” on page 592), **putchar**, **putwchar** and **tigetflag** (“**tigetflag**, **tigetnum**, **tigetstr**, or **tparm** Subroutine” on page 588) subroutines.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Setting Video Attributes and Curses Options in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

douupdate, 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.

Implementation Specifics

These subroutines are part of Base Operating System (BOS) Runtime.

Related Information

The **clearok** (“clearok, idlok, leaveok, scrollok, setscreg or wsetscreg Subroutine” on page 476) subroutine.

Curses Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

List of Curses Subroutines in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Manipulating Window Data with Curses in *AIX 5L Version 5.1 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, N
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 $(1 + (N-1) * \text{abs}(\text{INCX}))$; unchanged on exit.
INCX On entry, *INCX* specifies the increment for the elements of *X*; unchanged on exit.
Y Vector of dimension at least $(1 + (N-1) * \text{abs}(\text{INCY}))$; 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, N
COMPLEX X(*), Y(*)
```

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 $(1 + (N-1) * \text{abs}(\text{INCX}))$; unchanged on exit.
INCX On entry, *INCX* specifies the increment for the elements of *X*; unchanged on exit.
Y Vector of dimension at least $(1 + (N-1) * \text{abs}(\text{INCY}))$; unchanged on exit.
INCY On entry, *INCY* specifies the increment for the elements of *Y*; unchanged on exit.

Error Codes

For values of $N \leq 0$, 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 $(1 + (N-1) * \text{abs}(\text{INCX}))$; unchanged on exit.
INCX On entry, *INCX* specifies the increment for the elements of *X*; unchanged on exit.
Y Vector of dimension at least $(1 + (N-1) * \text{abs}(\text{INCY}))$; unchanged on exit.
INCY On entry, *INCY* specifies the increment for the elements of *Y*; unchanged on exit.

Error Codes

For values of $N \leq 0$, 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**)

FORTTRAN 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:

$$Y = A * X + Y$$

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 $(1 + (N-1) * \text{abs}(\text{INCX}))$; unchanged on exit.
- INCX* On entry, *INCX* specifies the increment for the elements of *X*; unchanged on exit.
- Y* Vector of dimension at least $(1 + (N-1) * \text{abs}(\text{INCY}))$; the result is returned in vector *Y*.
- INCY* On entry, *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:

$$a = \frac{A}{|A| + |B|}, \quad b = \frac{B}{|A| + |B|}$$

$$r = \begin{cases} a & \text{if } |A| > |B| \\ b & \text{if } |B| \geq |A| \end{cases} \quad r = \sqrt{a^2 + b^2}^{1/2}$$

$$C = \begin{cases} A/r & \text{if } r \text{ not} = 0 \\ 1 & \text{if } r = 0 \end{cases} \quad S = \begin{cases} B/r & \text{if } r \text{ not} = 0 \\ 0 & \text{if } r = 0 \end{cases}$$

The numbers *C*, *S*, and *r* then satisfy the matrix equation:

$$\begin{bmatrix} C & S \\ -S & C \end{bmatrix} \cdot \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

The subroutines also compute:

$$z = \begin{cases} S & \text{if } |A| > |B|, \\ 1/C & \text{if } |B| \geq |A| \text{ and } C \text{ not} = 0, \\ 1 & \text{if } C = 0. \end{cases}$$

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:

$$\begin{array}{|c|} \hline X \\ \hline i \\ \hline Y \\ \hline i \\ \hline \end{array} := \begin{array}{|cc|} \hline C & S \\ \hline -S & C \\ \hline \end{array} \cdot \begin{array}{|c|} \hline X \\ \hline i \\ \hline Y \\ \hline i \\ \hline \end{array} \quad \text{for } i = 1, \dots, N.$$

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 $(1 + (N-1) * \text{abs}(INCX))$; unchanged on exit.
- $INCX$ On entry, $INCX$ specifies the increment for the elements of X ; unchanged on exit.
- Y Vector of dimension at least $(1 + (N-1) * \text{abs}(INCY))$; 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 $(1 + (N-1) * \text{abs}(\text{INCX}))$; unchanged on exit.
- INCX* On entry, *INCX* specifies the increment for the elements of *X*; unchanged on exit.
- Y* Vector of dimension at least $(1 + (N-1) * \text{abs}(\text{INCY}))$ 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(*)
```

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**)

FORTTRAN Syntax

```
REAL FUNCTION SNRM2( $N,X,INCX$ )  
INTEGER  $INCX,N$   
REAL  $X(*)$   
  
DOUBLE PRECISION FUNCTION DNRM2( $N,X,INCX$ )  
INTEGER  $INCX,N$   
DOUBLE PRECISION  $X(*)$   
  
REAL FUNCTION SCNRM2( $N,X,INCX$ )  
INTEGER  $INCX,N$   
COMPLEX  $X(*)$   
  
DOUBLE PRECISION FUNCTION DZNRM2( $N,X,INCX$ )  
INTEGER  $INCX,N$   
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

```
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 $(1 + (N-1) * \text{abs}(INCX))$; 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* <= 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 SSCAL(N,A,X,INCX)
INTEGER INCX, N
REAL A
REAL X(*)
SUBROUTINE DSCAL(N,A,X,INCX)
INTEGER INCX,N
DOUBLE PRECISION A
DOUBLE PRECISION X(*)
SUBROUTINE CSSCAL(N,A,X,INCX)
INTEGER INCX,N
REAL A
COMPLEX X(*)
SUBROUTINE CSCAL
INTEGER INCX,N
COMPLEX A
COMPLEX X(*)
SUBROUTINE ZDSCAL
INTEGER INCX,N
DOUBLE PRECISION A
COMPLEX*16 X(*)
SUBROUTINE ZSCAL(
INTEGER INCX,N
COMPLEX*16 A
COMPLEX*16 X(*)
```

Description

The **SSCAL**, **DSCAL**, **CSSCAL**, **CSCAL**, **ZDSCAL**, or **ZSCAL** subroutine scales a vector by a constant:

$X := X * A$

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 $(1 + (N-1) * \text{abs}(\text{INCX}))$; 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* ≤ 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 $(1 + (N-1) * \text{abs}(\text{INCX}))$; 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 *N, INCX, INCY*

REAL *B, X*(*), *Y*(*)

Purpose

The **SDSDOT** function computes the sum of constant *B* and dot product of vectors *X* and *Y*.

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 $(1 + (N-1) * \text{abs}(\text{INCX}))$; 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 $(1 + (N-1) * \text{abs}(INCY))$; 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.

Implementation Specifics

Computation is performed in double precision.

SROTM or DROTM Subroutine

Purpose

Applies the modified Givens transformation.

Library

BLAS Library (*libblas.a*)

FORTRAN Syntax

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:

$$\begin{bmatrix} x \\ y \end{bmatrix} := H * \begin{bmatrix} x \\ y \end{bmatrix}$$

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)
  
```

Parameters

<i>N</i>	On entry, <i>N</i> specifies the number of elements in <i>X</i> and <i>Y</i> ; unchanged on exit.
<i>X</i>	Vector of dimension at least $(1 + (N-1) * \text{abs}(INCX))$; on exit, modified as described above.
<i>INCX</i>	On entry, <i>INCX</i> specifies the increment for the elements of <i>X</i> ; <i>INCX</i> must be greater than 0; unchanged on exit.
<i>Y</i>	Vector of dimension at least $(1 + (N-1) * \text{abs}(INCY))$; on exit, modified as described above.
<i>INCY</i>	On entry, <i>INCY</i> specifies the increment for the elements of <i>Y</i> ; <i>INCY</i> must be greater than 0; unchanged on exit.
<i>PARAM</i>	Vector of dimension (5); on entry, must be set as described above. Specifically, <i>PARAM</i> (1) is a flag and must have value of either 0.0, -1.0, 1.0, or 2.0; unchanged on exit.

Implementation Specifics

If $N \leq 0$ or *H* is an identity matrix, the subroutines return immediately.

Related information

The **SROTMG** or **DROTMG** ("SROTMG or DROTMG Subroutine") subroutine builds the *PARAM* array prior to use by the **SROTM** or **DROTM** 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:

$$\begin{bmatrix} a1 \\ a2 \end{bmatrix} = \begin{bmatrix} \text{sqrt}(D1) & 0 \\ 0 & \text{sqrt}(D2) \end{bmatrix} \begin{bmatrix} X1 \\ X2 \end{bmatrix}$$

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: $PARAM(1) = -2.0$
 $H = I$ (Identity matrix)

Note: Locations in *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.
PARAM Vector of dimension (5); values on entry are unused; modified on exit as described above.

Related Information

The **SROTM** and **DROTM** (“SROTM or DROTM Subroutine” on page 615) 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(TRANS, M, N, ALPHA, A, LDA, X,
  INCX, BETA, Y, INCY)
REAL ALPHA, BETA
INTEGER INCX, INCY, LDA, M, N
CHARACTER*1 TRANS
REAL A(LDA,*), X(*), Y(*)

SUBROUTINE DGEMV(TRANS, M, N, ALPHA, A, LDA, X,
  INCX, BETA, Y, INCY)
DOUBLE PRECISION ALPHA,BETA
INTEGER INCX, INCY, LDA, M, N
CHARACTER*1 TRANS
DOUBLE PRECISION A(LDA,*), X(*), Y(*)

SUBROUTINE CGEMV(TRANS, M, N, ALPHA, A, LDA, X,
  INCX, BETA, Y, INCY)
COMPLEX ALPHA,BETA
INTEGER INCX, INCY, LDA, M, N
CHARACTER*1 TRANS
COMPLEX A(LDA,*), X(*), Y(*)

```

```

SUBROUTINE ZGEMV(TRANS, M, N, ALPHA, A, LDA, X,
INCX, BETA, Y, INCY)
COMPLEX*16 ALPHA,BETA
INTEGER INCX,INCY,LDA,M,N
CHARACTER*1 TRANS
COMPLEX*16 A(LDA,*), X(*), Y(*)

```

Description

The **SGEMV**, **DGEMV**, **CGEMV**, or **ZGEMV** subroutine performs one of the following matrix-vector operations:

$$y := \alpha * A * x + \beta * y$$

OR

$$y := \alpha * A' * x + \beta * y$$

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

Parameters

TRANS On entry, *TRANS* specifies the operation to be performed as follows:

TRANS = 'N' or 'n'

$$y := \alpha * A * x + \beta * y$$

TRANS = 'T' or 't'

$$y := \alpha * A' * x + \beta * y$$

TRANS = 'C' or 'c'

$$y := \alpha * A' * x + \beta * y$$

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.

ALPHA On entry, *ALPHA* specifies the scalar alpha; 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; 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, *M*); unchanged on exit.

X A vector of dimension at least (1 + (*N*-1) * abs(*INCX*)) when *TRANS* = 'N' or 'n', otherwise, at least (1 + (*M*-1) * 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; when *BETA* is supplied as 0, *Y* need not be set on input; unchanged on exit.

Y A vector of dimension at least 1 + (*M*-1) * abs(*INCY*) when *TRANS* = 'N' or 'n', otherwise at least (1 + (*N*-1) * abs(*INCY*)); on entry, with *BETA* nonzero, 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.

SGBMV, DGBMV, CGBMV, or ZGBMV Subroutine

Purpose

Performs matrix-vector operations with general banded matrices.

Library

BLAS Library (**libblas.a**)

FORTRAN Syntax

```
SUBROUTINE SGBMV(TRANS, M, N, KL, KU, ALPHA, A, LDA,  
X, INCX, BETA, Y, INCY)  
REAL ALPHA, BETA  
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:

$y := \alpha * A * x + \beta * y$

OR

$y := \alpha * A' * x + \beta * y$

where α and β 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'

$y := \alpha * A * x + \beta * y$

TRANS = 'T' or 't'

$y := \alpha * A' * x + \beta * y$

TRANS = 'C' or 'c'

$y := \alpha * A' * x + \beta * y$

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 $0 \leq KL$; unchanged on exit.

KU On entry, *KU* specifies the number of superdiagonals of the matrix *A*; *KU* must satisfy $0 \leq KU$; unchanged on exit.

ALPHA On entry, *ALPHA* specifies the scalar alpha; unchanged on exit.

A 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:

```

DO 20, J = 1, N
  K = KU + 1 - J
  DO 10, I = MAX( 1, J - KU ), MIN( M, J + KL )
    A( K + I, J ) = matrix( I, J )
  10 CONTINUE
  20 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) * abs(*INCX*)) when *TRANS* = 'N' or 'n', otherwise, at least (1 + (*M*-1) * 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; 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) * abs(*INCY*)) when *TRANS* = 'N' or 'n', otherwise, at least (1 + (*N*-1) * 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 (**libblas.a**)

FORTRAN Syntax

```

SUBROUTINE CHEMV(UPLO, N, ALPHA, A, LDA,
X, INCX, BETA, Y, INCY)

```

```

COMPLEX ALPHA, BETA

```

```

INTEGER INCX, INCY, LDA, N

```

```

CHARACTER*1 UPLO

```

```

COMPLEX A(LDA,*), X(*), Y(*)

```

```

SUBROUTINE ZHEMV(UPLO, N, ALPHA, A, LDA,
X, INCX, BETA, Y, INCY)

```

```

COMPLEX*16 ALPHA, BETA

```

```

INTEGER INCX, INCY, LDA, N

```

```

CHARACTER*1 UPLO

```

```

COMPLEX*16 A(LDA,*), X(*), Y(*)

```

Description

The **CHEMV** or **ZHEMV** subroutine performs the matrix-vector operation:

$$y := \alpha * A * x + \beta * y$$

where alpha and beta are scalars, x and y are *N* element vectors and *A* is an *N* by *N* Hermitian matrix.

Parameters

<i>UPLO</i>	On entry, <i>UPLO</i> specifies whether the upper or lower triangular part of the array <i>A</i> is to be referenced as follows: <i>UPLO</i> = 'U' or 'u' Only the upper triangular part of <i>A</i> is to be referenced; unchanged on exit. <i>UPLO</i> = 'L' or 'l' Only the lower triangular part of <i>A</i> is to be referenced; unchanged on exit.
<i>N</i>	On entry, <i>N</i> specifies the order of the matrix <i>A</i> ; <i>N</i> must be at least 0; unchanged on exit.
<i>ALPHA</i>	On entry, <i>ALPHA</i> specifies the scalar alpha; unchanged on exit.
<i>A</i>	An array of dimension (<i>LDA</i> , <i>N</i>); on entry with <i>UPLO</i> = 'U' or 'u', the leading <i>N</i> by <i>N</i> upper triangular part of the array <i>A</i> must contain the upper triangular part of the Hermitian matrix and the strictly lower triangular part of <i>A</i> is not referenced; on entry with <i>UPLO</i> = 'L' or 'l', the leading <i>N</i> by <i>N</i> lower triangular part of the array <i>A</i> must contain the lower triangular part of the Hermitian matrix and the strictly upper triangular part of <i>A</i> is not referenced. The imaginary parts of the diagonal elements need not be set and are assumed to be 0; unchanged on exit.
<i>LDA</i>	On entry, <i>LDA</i> specifies the first dimension of <i>A</i> as declared in the calling (sub) program; <i>LDA</i> must be at least max(1, <i>N</i>); unchanged on exit.
<i>X</i>	A vector of dimension at least (1 + (<i>N</i> -1) * abs(<i>INCX</i>)); on entry, the incremented array <i>X</i> must contain the <i>N</i> element vector <i>x</i> ; unchanged on exit.
<i>INCX</i>	On entry, <i>INCX</i> specifies the increment for the elements of <i>X</i> ; <i>INCX</i> must not be 0; unchanged on exit.
<i>BETA</i>	On entry, <i>BETA</i> specifies the scalar beta; when <i>BETA</i> is supplied as 0 then <i>Y</i> need not be set on input; unchanged on exit.
<i>Y</i>	A vector of dimension at least (1 + (<i>N</i> -1) * abs(<i>INCX</i>)); on entry, the incremented array <i>Y</i> must contain the <i>N</i> element vector <i>y</i> ; on exit, <i>Y</i> is overwritten by the updated vector <i>y</i> .
<i>INCY</i>	On entry, <i>INCY</i> specifies the increment for the elements of <i>Y</i> ; <i>INCY</i> 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

```
SUBROUTINE CHBMV(UPLO, N, K, ALPHA, A, LDA,  
X, INCX, BETA, Y, INCY)
```

```
COMPLEX ALPHA, BETA
```

```
INTEGER INCX, INCY, K, LDA, N
```

```
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:

$y := \alpha * A * x + \beta * y$

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 ($1 + (N-1) * \text{abs}(INCX)$); 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 ($1 + (N-1) * \text{abs}(INCY)$); 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:

$$y := \alpha * A * x + \beta * y$$

where α and β 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, **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 α ; unchanged on exit.

AP A vector of dimension at least $((N * (N+1)) / 2)$; 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 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. 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 $(1 + (N-1) * \text{abs}(\text{INCX}))$; 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 β ; when **BETA** is supplied as 0 then **Y** need not be set on input; unchanged on exit.

Y A vector of dimension at least $(1 + (N-1) * \text{abs}(\text{INCY}))$; on entry, the incremented array *Y* must contain the *N* element vector *y*; on exit, *Y* is overwritten by the updated vector *y*.
INCY On entry, *INCY* specifies the increment for the elements of *Y*; *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

```
SUBROUTINE SSYMV(UPLO, N, ALPHA, A, LDA,  
X, INCX, BETA, Y, INCY)
```

```
REAL ALPHA, BETA
```

```
INTEGER INCX, 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 INCX, INCY, LDA, N
```

```
CHARACTER*1 UPLO
```

```
DOUBLE PRECISION A(LDA,*), X(*), Y(*)
```

Description

The **SSYMV** or **DSYMV** subroutine performs the matrix-vector operation:

$$y := \alpha * A * x + \beta * y$$

where α and β are scalars, 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 = '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.

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 α ; 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 symmetric matrix; 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 symmetric matrix; the strictly upper triangular part of *A* is not referenced; 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.

<i>X</i>	A vector of dimension at least $(1 + (N-1) * \text{abs}(INCX))$; on entry, the incremented array <i>X</i> must contain the <i>N</i> element vector <i>x</i> ; unchanged on exit.
<i>INCX</i>	On entry, <i>INCX</i> specifies the increment for the elements of <i>X</i> ; <i>INCX</i> must not be 0; unchanged on exit.
<i>BETA</i>	On entry, <i>BETA</i> specifies the scalar beta; when <i>BETA</i> is supplied as 0 then <i>Y</i> need not be set on input; unchanged on exit.
<i>Y</i>	A vector of dimension at least $(1 + (N-1) * \text{abs}(INCY))$; on entry, the incremented array <i>Y</i> must contain the <i>N</i> element vector <i>y</i> ; on exit, <i>Y</i> is overwritten by the updated vector <i>y</i> .
<i>INCY</i>	On entry, <i>INCY</i> specifies the increment for the elements of <i>Y</i> ; <i>INCY</i> 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:

$$y := \text{alpha} * A * x + \text{beta} * y$$

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 = '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 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:

```

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 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:

```

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

```

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) * 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) * 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(*)

```

```

SUBROUTINE DSPMV(UPLO, N, ALPHA, AP, X,
INCX, BETA, Y, INCY)
DOUBLE PRECISION ALPHA,BETA
INTEGER INCX,INCY,N
CHARACTER*1 UPLO
DOUBLE PRECISION AP(*), X(*), Y(*)

```

Description

The **SSPMV** or **DSPMV** subroutine performs the matrix-vector operation:

$$y := \alpha * A * x + \beta * y$$

where α and β 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*.
- N* 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 α ; unchanged on exit.
- AP* A vector of dimension at least $((N * (N+1)) / 2)$; 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 $(1 + (N-1) * \text{abs}(INCX))$; 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 β ; when *BETA* is supplied as 0 then *Y* need not be set on input; unchanged on exit.
- Y* A vector of dimension at least $(1 + (N-1) * \text{abs}(INCY))$; 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)

```

```

INTEGER INCX, LDA, N
CHARACTER*1 DIAG, TRANS, UPLO
REAL A(LDA,*), X(*)
SUBROUTINE DTRMV(UPLO, TRANS, DIAG, N,
A, LDA, X, INCX)
INTEGER INCX,LDA,N
CHARACTER*1 DIAG,TRANS,UPLO
DOUBLE PRECISION A(LDA,*), X(*)
SUBROUTINE CTRMV(UPLO, TRANS, DIAG, N,
A, LDA, X, INCX)
INTEGER INCX,LDA,N
CHARACTER*1 DIAG,TRANS,UPLO
COMPLEX A(LDA,*), X(*)
SUBROUTINE ZTRMV(UPLO, TRANS, DIAG, N,
A, LDA, X, INCX)
INTEGER INCX,LDA,N
CHARACTER*1 DIAG,TRANS,UPLO
COMPLEX*16 A(LDA,*),X(*)

```

Description

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.

<i>N</i>	On entry, <i>N</i> specifies the order of the matrix <i>A</i> ; <i>N</i> must be at least 0; unchanged on exit.
<i>A</i>	An array of dimension (<i>LDA</i> , <i>N</i>); on entry with <i>UPLO</i> = 'U' or 'u', the leading <i>N</i> by <i>N</i> upper triangular part of the array <i>A</i> must contain the upper triangular matrix and the strictly lower triangular part of <i>A</i> is not referenced; on entry with <i>UPLO</i> = 'L' or 'l', the leading <i>N</i> by <i>N</i> lower triangular part of the array <i>A</i> must contain the lower triangular matrix and the strictly upper triangular part of <i>A</i> is not referenced. When <i>DIAG</i> = 'U' or 'u', the diagonal elements of <i>A</i> are not referenced, but are assumed to be unity; unchanged on exit.
<i>LDA</i>	On entry, <i>LDA</i> specifies the first dimension of <i>A</i> as declared in the calling (sub) program. <i>LDA</i> must be at least max(1, <i>N</i>); unchanged on exit.
<i>X</i>	A vector of dimension at least (1 + (<i>N</i> -1) * abs(<i>INCX</i>)). On entry, the incremented array <i>X</i> must contain the <i>N</i> element vector <i>x</i> ; on exit, <i>X</i> is overwritten with the transformed vector <i>x</i> .
<i>INCX</i>	On entry, <i>INCX</i> specifies the increment for the elements of <i>X</i> ; <i>INCX</i> 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

```
SUBROUTINE STBMV(UPLO, TRANS, DIAG, N,  
K, A, LDA, X, INCX)
```

```
INTEGER INCX, K, LDA, N
```

```
CHARACTER*1 DIAG, TRANS, UPLO
```

```
REAL A(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 A(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 * 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 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'

$x := A * x$

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.

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 $0 \leq K$; 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:

```
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
```

```
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
```

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:

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) * 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 * 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, 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'
 $x := A' * x$
 - 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.
- 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 $((N * (N+1)) / 2)$. 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 $(1 + (N-1) * \text{abs}(INCX))$; 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.

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.

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.

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.

<i>A</i>	An array of dimension (<i>LDA</i> , <i>N</i>); on entry with <i>UPLO</i> = 'U' or 'u', the leading <i>N</i> by <i>N</i> upper triangular part of the array <i>A</i> must contain the upper triangular matrix and the strictly lower triangular part of <i>A</i> is not referenced. On entry with <i>UPLO</i> = 'L' or 'l', the leading <i>N</i> by <i>N</i> lower triangular part of the array <i>A</i> must contain the lower triangular matrix and the strictly upper triangular part of <i>A</i> is not referenced. When <i>DIAG</i> = 'U' or 'u', the diagonal elements of <i>A</i> are not referenced, but are assumed to be unity; unchanged on exit.
<i>LDA</i>	On entry, <i>LDA</i> specifies the first dimension of <i>A</i> as declared in the calling (sub) program; <i>LDA</i> must be at least max(1, <i>N</i>); unchanged on exit.
<i>X</i>	A vector of dimension at least (1 + (<i>N</i> -1) * abs(<i>INCX</i>)); on entry, the incremented array <i>X</i> must contain the <i>N</i> element right-hand side vector <i>b</i> ; on exit, <i>X</i> is overwritten with the solution vector <i>x</i> .
<i>INCX</i>	On entry, <i>INCX</i> specifies the increment for the elements of <i>X</i> ; <i>INCX</i> must not be 0; unchanged on exit.

Implementation Specifics

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

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:

$$A * x = b$$

OR

$$A^T * x = b$$

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.

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.

DIAG On entry, *DIAG* specifies whether 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 $0 \leq K < N$; 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:

```
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:

```
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) * 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.

Implementation Specifics

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

STPSV, DTPSV, CTPSV, or ZTPSV Subroutine

Purpose

Solves systems of equations.

Library

BLAS Library (**libblas.a**)

FORTRAN Syntax

```
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**, **CTPSV**, or **ZTPSV** 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, 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 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.

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 $((N * (N+1)) / 2)$; 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. Before 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 $(1 + (N-1) * \text{abs}(INCX))$; 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.

Implementation Specifics

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

SGER or DGER Subroutine

Purpose

Performs the rank 1 operation.

Library

BLAS Library (**libblas.a**)

FORTRAN Syntax

```
SUBROUTINE SGER(M, N, ALPHA, X,
  INCX, Y, INCY, A, LDA)
```

```
REAL ALPHA
```

```
INTEGER INCX, INCY, LDA, M, N
```

```
REAL A(LDA,*), X(*), Y(*)
```

```
SUBROUTINE DGER(M, N, ALPHA, X,
  INCX, Y, INCY, A, LDA)
```

```
DOUBLE PRECISION ALPHA
```

```
INTEGER INCX, INCY, LDA, M, N
```

```
DOUBLE PRECISION A(LDA,*), X(*), Y(*)
```

Description

The **SGER** or **DGER** subroutine performs the rank 1 operation:

$$A := \alpha * x * y' + A$$

where α 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 $(1 + (M-1) * \text{abs}(INCX))$; on entry, the incremented array *X* must contain the *M* 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 $(1 + (N-1) * \text{abs}(INCY))$; 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, 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.

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(*)
```

```
SUBROUTINE ZGERU
```

```
COMPLEX*16 ALPHA
```

```
INTEGER INCX, INCY, LDA, M, N
```

```
COMPLEX*16 A(LDA,*), X(*), Y(*)
```

Description

The **CGERU** or **ZGERU** subroutine performs the rank 1 operation:

$$A := \alpha * x * y' + A$$

where α 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 α ; unchanged on exit.
- X* A vector of dimension at least $(1 + (M-1) * \text{abs}(INCX))$; on entry, the incremented array *X* must contain the M 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 $(1 + (N-1) * \text{abs}(INCY))$; 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, 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:

$$A := \alpha * x * \text{conjg}(y^T) + A$$

where α is a scalar, x is an M element vector, y is an N element vector and A is an M by N matrix.

Parameters

<i>M</i>	On entry, <i>M</i> specifies the number of rows of the matrix <i>A</i> ; <i>M</i> must be at least 0; unchanged on exit.
<i>N</i>	On entry, <i>N</i> specifies the number of columns of the matrix <i>A</i> ; <i>N</i> must be at least 0; unchanged on exit.
<i>ALPHA</i>	On entry, <i>ALPHA</i> specifies the scalar α ; unchanged on exit.
<i>X</i>	A vector of dimension at least $(1 + (M-1) * \text{abs}(INCX))$; on entry, the incremented array <i>X</i> must contain the <i>M</i> element vector x ; unchanged on exit.
<i>INCX</i>	On entry, <i>INCX</i> specifies the increment for the elements of <i>X</i> ; <i>INCX</i> must not be 0; unchanged on exit.
<i>Y</i>	A vector of dimension at least $(1 + (N-1) * \text{abs}(INCY))$; on entry, the incremented array <i>Y</i> must contain the <i>N</i> element vector y ; unchanged on exit.
<i>INCY</i>	On entry, <i>INCY</i> specifies the increment for the elements of <i>Y</i> ; <i>INCY</i> must not be 0; unchanged on exit.
<i>A</i>	An array of dimension (LDA, N) ; on entry, the leading <i>M</i> by <i>N</i> part of the array <i>A</i> must contain the matrix of coefficients; on exit, <i>A</i> is overwritten by the updated matrix.
<i>LDA</i>	On entry, <i>LDA</i> specifies the first dimension of <i>A</i> as declared in the calling (sub) program; <i>LDA</i> 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(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:

$$A := \alpha * x * \text{conjg}(x^T) + A$$

where α 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:

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.

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 α ; unchanged on exit.

X A vector of dimension at least $(1 + (N-1) * \text{abs}(\text{INCX}))$; 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 *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.

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

```

```

INTEGER INCX, N
CHARACTER*1 UPLO
COMPLEX AP(*), X(*)
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:

$$A := \alpha * x * \text{conjg}(x') + A$$

where α 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 α ; unchanged on exit.

X 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 ; 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 $(N * (N+1) / 2)$; 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)

```

```

COMPLEX ALPHA
INTEGER INCX, INCY, LDA, N
CHARACTER*1 UPLO
COMPLEX A(LDA,*), X(*), Y(*)
SUBROUTINE ZHER2(UPLO, N, ALPHA,
X, INCX, Y, INCY, A, LDA)
COMPLEX*16 ALPHA
INTEGER INCX, INCY, LDA, N
CHARACTER*1 UPLO
COMPLEX*16 A(LDA,*), X(*), Y(*)

```

Description

The **CHER2** or **ZHER2** subroutine performs the Hermitian rank 2 operation:

$$A := \alpha * x * \text{conjg}(y') + \text{conjg}(\alpha) * y * \text{conjugate}(x') + A$$

where α is a scalar, 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.

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 α ; unchanged on exit.

X A vector of dimension at least $(1 + (N-1) * \text{abs}(INCX))$; 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 $(1 + (N-1) * \text{abs}(INCY))$; 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:

$$A := \alpha * x * \text{conjg}(y') + \text{conjg}(\alpha) * y * \text{conjg}(x') + A$$

where α 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, *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 α ; unchanged on exit.
- X* A vector of dimension at least $(1 + (N-1) * \text{abs}(\text{INCX}))$; 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 $(1 + (N-1) * \text{abs}(\text{INCY}))$; 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.
- AP* A vector of dimension at least $((N * (N+1)) / 2)$; 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.

SSYR or DSYR Subroutine

Purpose

Performs the symmetric rank 1 operation.

Library

BLAS Library (**libblas.a**)

FORTTRAN 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:

$$A := \alpha x x' + A$$

where α is a real scalar, x is an N element vector 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 = '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.

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 α ; unchanged on exit.

X A vector of dimension at least $(1 + (N-1) * \text{abs}(\text{INCX}))$; 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 **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 **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, **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.

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:

$$A := \alpha * x * x' + A$$

where α 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 α ; unchanged on exit.

X A vector of dimension at least $(1 + (N-1) * \text{abs}(\text{INCX}))$; 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 $((N * (N+1)) / 2)$; 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.

SSYR2 or DSYR2 Subroutine

Purpose

Performs the symmetric rank 2 operation.

Library

BLAS Library (**libblas.a**)

FORTRAN Syntax

```
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:

$$A := \alpha * x * x' + \alpha * y * y' + A$$

where α is a scalar, x and y are N element vectors and A is an N by N symmetric matrix.

Parameters

<i>UPLO</i>	On entry, <i>UPLO</i> specifies whether the upper or lower triangular part of the array <i>A</i> is to be referenced as follows: <i>UPLO</i> = 'U' or 'u' Only the upper triangular part of <i>A</i> is to be referenced. <i>UPLO</i> = 'L' or 'l' Only the lower triangular part of <i>A</i> is to be referenced. Unchanged on exit.
<i>N</i>	On entry, <i>N</i> specifies the order of the matrix <i>A</i> ; <i>N</i> must be at least 0; unchanged on exit.
<i>ALPHA</i>	On entry, <i>ALPHA</i> specifies the scalar α ; unchanged on exit.
<i>X</i>	A vector of dimension at least $(1 + (N-1) * \text{abs}(\text{INCX}))$; on entry, the incremented array <i>X</i> must contain the N element vector x ; unchanged on exit.
<i>INCX</i>	On entry, <i>INCX</i> specifies the increment for the elements of <i>X</i> ; <i>INCX</i> must not be 0; unchanged on exit.
<i>Y</i>	A vector of dimension at least $(1 + (N-1) * \text{abs}(\text{INCY}))$; on entry, the incremented array <i>Y</i> must contain the N element vector y ; unchanged on exit.
<i>INCY</i>	On entry, <i>INCY</i> specifies the increment for the elements of <i>Y</i> ; <i>INCY</i> must not be 0; unchanged on exit.
<i>A</i>	An array of dimension (LDA, N) ; on entry with <i>UPLO</i> = 'U' or 'u', the leading N by N upper triangular part of the array <i>A</i> must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of <i>A</i> is not referenced. On exit, the upper triangular part of the array <i>A</i> is overwritten by the upper triangular part of the updated matrix. On entry with <i>UPLO</i> = 'L' or 'l', the leading N by N lower triangular part of the array <i>A</i> must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of <i>A</i> is not referenced. On exit, the lower triangular part of the array <i>A</i> is overwritten by the lower triangular part of the updated matrix.
<i>LDA</i>	On entry, <i>LDA</i> specifies the first dimension of <i>A</i> as declared in the calling (sub) program; <i>LDA</i> 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 α 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 α ; unchanged on exit.

X A vector of dimension at least $(1 + (N-1) * \text{abs}(\text{INCX}))$; 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 $(1 + (N-1) * \text{abs}(\text{INCY}))$; 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.

AP A vector of dimension at least $((N * (N+1))/2)$; 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:

$$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$$

where *op*(*X*) is one of *op*(*X*) = *X* or *op*(*X*) = *X*' , *alpha* and *beta* are scalars, and *A*, *B* and *C* are matrices, with *op*(*A*) an *M* by *K* matrix, *op*(*B*) a *K* by *N* matrix and *C* an *M* by *N* matrix.

Parameters

<i>TRANSA</i>	On entry, <i>TRANSA</i> specifies the form of $op(A)$ to be used in the matrix multiplication as follows: <i>TRANSA</i> = 'N' or 'n' $op(A) = A$ <i>TRANSA</i> = 'T' or 't' $op(A) = A'$ <i>TRANSA</i> = 'C' or 'c' $op(A) = A'$ Unchanged on exit.
<i>TRANSB</i>	On entry, <i>TRANSB</i> specifies the form of $op(B)$ to be used in the matrix multiplication as follows: <i>TRANSB</i> = 'N' or 'n' $op(B) = B$ <i>TRANSB</i> = 'T' or 't' $op(B) = B'$ <i>TRANSB</i> = 'C' or 'c' $op(B) = B'$ Unchanged on exit.
<i>M</i>	On entry, <i>M</i> specifies the number of rows of the matrix $op(A)$ and of the matrix <i>C</i> ; <i>M</i> must be at least 0; unchanged on exit.
<i>N</i>	On entry, <i>N</i> specifies the number of columns of the matrix $op(B)$ and the number of columns of the matrix <i>C</i> ; <i>N</i> must be at least 0; unchanged on exit.
<i>K</i>	On entry, <i>K</i> specifies the number of columns of the matrix $op(A)$ and the number of rows of the matrix $op(B)$; <i>K</i> must be at least 0; unchanged on exit.
<i>ALPHA</i>	On entry, <i>ALPHA</i> specifies the scalar alpha; unchanged on exit.
<i>A</i>	An array of dimension (<i>LDA</i> , <i>KA</i>), where <i>KA</i> is <i>K</i> when <i>TRANSA</i> = 'N' or 'n', and is <i>M</i> otherwise; on entry with <i>TRANSA</i> = 'N' or 'n', the leading <i>M</i> by <i>K</i> part of the array <i>A</i> must contain the matrix <i>A</i> , otherwise the leading <i>K</i> by <i>M</i> part of the array <i>A</i> must contain the matrix <i>A</i> ; unchanged on exit.
<i>LDA</i>	On entry, <i>LDA</i> specifies the first dimension of <i>A</i> as declared in the calling (sub) program. When <i>TRANSA</i> = 'N' or 'n' then <i>LDA</i> must be at least $\max(1, M)$, otherwise <i>LDA</i> must be at least $\max(1, K)$; unchanged on exit.
<i>B</i>	An array of dimension (<i>LDB</i> , <i>KB</i>) where <i>KB</i> is <i>N</i> when <i>TRANSB</i> = 'N' or 'n', and is <i>K</i> otherwise; on entry with <i>TRANSB</i> = 'N' or 'n', the leading <i>K</i> by <i>N</i> part of the array <i>B</i> must contain the matrix <i>B</i> , otherwise the leading <i>N</i> by <i>K</i> part of the array <i>B</i> must contain the matrix <i>B</i> ; unchanged on exit.
<i>LDB</i>	On entry, <i>LDB</i> specifies the first dimension of <i>B</i> as declared in the calling (sub) program. When <i>TRANSB</i> = 'N' or 'n' then <i>LDB</i> must be at least $\max(1, K)$, otherwise <i>LDB</i> must be at least $\max(1, N)$; unchanged on exit.
<i>BETA</i>	On entry, <i>BETA</i> specifies the scalar beta. When <i>BETA</i> is supplied as 0 then <i>C</i> need not be set on input; unchanged on exit.
<i>C</i>	An array of dimension (<i>LDC</i> , <i>N</i>); on entry, the leading <i>M</i> by <i>N</i> part of the array <i>C</i> must contain the matrix <i>C</i> , except when beta is 0, in which case <i>C</i> need not be set on entry; on exit, the array <i>C</i> is overwritten by the <i>M</i> by <i>N</i> matrix ($\alpha * op(A) * op(B) + \beta * C$).
<i>LDC</i>	On entry, <i>LDC</i> specifies the first dimension of <i>C</i> as declared in the calling (sub) program; <i>LDC</i> must be at least $\max(1, M)$; 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:

$$C := \alpha * A * B + \beta * C$$

OR

$$C := \alpha * B * A + \beta * C$$

where α and β 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'

$$C := \alpha * A * B + \beta * C$$

SIDE = 'R' or 'r'

$$C := \alpha * B * A + \beta * C$$

Unchanged on exit.

UPLO On entry, *UPLO* specifies whether the upper or lower triangular part of the symmetric matrix *A* is to be referenced as follows:

UPLO = 'U' or 'u'
Only the upper triangular part of the symmetric matrix is to be referenced.

UPLO = 'L' or 'l'
Only the lower triangular part of the symmetric 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, *ALPHA* 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 symmetric 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 symmetric 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 symmetric 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 symmetric 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 symmetric 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 symmetric matrix and the strictly upper triangular part of *A* is not referenced; 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.

BETA On entry, *BETA* specifies the scalar beta; when *BETA* is supplied as 0 then *C* need not be set on input; unchanged on exit.

C 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.

LDC 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.

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:

$C := \alpha * A * B + \beta * C$

OR

$C := \alpha * B * A + \beta * C$

where α and β 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'

$C := \alpha * A * B + \beta * C$

SIDE = 'R' or 'r'

$C := \alpha * B * A + \beta * C$

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, **ALPHA** specifies the scalar α ; 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.

<i>BETA</i>	On entry, <i>BETA</i> specifies the scalar beta. When <i>BETA</i> is supplied as 0 then <i>C</i> need not be set on input; unchanged on exit.
<i>C</i>	An array of dimension (<i>LDC</i> , <i>N</i>); on entry, the leading <i>M</i> by <i>N</i> part of the array <i>C</i> must contain the matrix <i>C</i> , except when beta is 0, in which case <i>C</i> need not be set on entry; on exit, the array <i>C</i> is overwritten by the <i>M</i> by <i>N</i> updated matrix.
<i>LDC</i>	On entry, <i>LDC</i> specifies the first dimension of <i>C</i> as declared in the calling (sub) program; <i>LDC</i> 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

```
SUBROUTINE SSYRK(UPLO, TRANS, N, K, ALPHA,
A, LDA, BETA, C, LDC)
```

```
CHARACTER*1 UPLO, TRANS
```

```
INTEGER N, 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:

$$C := \alpha * A * A' + \beta * C$$

OR

$$C := \alpha * A' * A + \beta * C$$

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:

$$C := \alpha * A * \text{conjg}(A') + \beta * C$$

OR

$$C := \alpha * \text{conjg}(A') * A + \beta * C$$

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'

$$C := \alpha * A * \text{conjg}(A') + \beta * C$$

TRANS = 'C' or 'c'

$$C := \alpha * \text{conjg}(A') * A + \beta * C$$

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* = '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 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.
- 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.

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 **SSYR2K**, **DSYR2K**, **CSYR2K**, or **ZSYR2K** subroutine performs one of the symmetric rank 2k operations:

$C := \alpha * A * B' + \alpha * B * A' + \beta * C$

OR

$C := \alpha * A' * B + \alpha * B' * A + \beta * C$

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'
 $C := \alpha * A * B + \alpha * B * A + \beta * C$
- TRANS* = 'T' or 't'
 $C := \alpha * A' * B + \alpha * B' * A + \beta * C$
- 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* = '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**)

FORTTRAN Syntax

SUBROUTINE **CHER2K**(*UPLO*, *TRANS*, *N*, *K*, *ALPHA*,
A, *LDA*, *B*, *LDB*, *C*, *LDC*)

CHARACTER*1 *UPLO*, *TRANS*

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*, *TRANS*

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 α and β are scalars with β 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 α ; unchanged on exit.

<i>A</i>	An array of dimension (<i>LDA</i> , <i>KA</i>), where <i>KA</i> is <i>K</i> when <i>TRANS</i> = 'N' or 'n', and is <i>N</i> otherwise; on entry with <i>TRANS</i> = 'N' or 'n', the leading <i>N</i> by <i>K</i> part of the array <i>A</i> must contain the matrix <i>A</i> , otherwise the leading <i>K</i> by <i>N</i> part of the array <i>A</i> must contain the matrix <i>A</i> ; unchanged on exit.
<i>LDA</i>	On entry, <i>LDA</i> specifies the first dimension of <i>A</i> as declared in the calling (sub) program. When <i>TRANS</i> = 'N' or 'n', <i>LDA</i> must be at least max(1, <i>N</i>); otherwise <i>LDA</i> must be at least max(1, <i>K</i>); unchanged on exit.
<i>B</i>	An array of dimension (<i>LDB</i> , <i>KB</i>), where <i>KB</i> is <i>K</i> when <i>TRANS</i> = 'N' or 'n', and is <i>N</i> otherwise; on entry with <i>TRANS</i> = 'N' or 'n', the leading <i>N</i> by <i>K</i> part of the array <i>B</i> must contain the matrix <i>B</i> , otherwise the leading <i>K</i> by <i>N</i> part of the array <i>B</i> must contain the matrix <i>B</i> ; unchanged on exit.
<i>LDB</i>	On entry, <i>LDB</i> specifies the first dimension of <i>B</i> as declared in the calling (sub) program. When <i>TRANS</i> = 'N' or 'n', <i>LDB</i> must be at least max(1, <i>N</i>); otherwise <i>LDB</i> must be at least max(1, <i>K</i>); unchanged on exit.
<i>BETA</i>	On entry, <i>BETA</i> specifies the scalar beta; unchanged on exit.
<i>C</i>	An array of dimension (<i>LDC</i> , <i>N</i>); on entry with <i>UPLO</i> = 'U' or 'u', the leading <i>N</i> by <i>N</i> upper triangular part of the array <i>C</i> must contain the upper triangular part of the Hermitian matrix and the strictly lower triangular part of <i>C</i> is not reference; on exit, the upper triangular part of the array <i>C</i> is overwritten by the upper triangular part of the updated matrix; on entry with <i>UPLO</i> = 'L' or 'l', the leading <i>N</i> by <i>N</i> lower triangular part of the array <i>C</i> must contain the lower triangular part of the Hermitian matrix and the strictly upper triangular part of <i>C</i> is not referenced; on exit, the lower triangular part of the array <i>C</i> 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.
<i>LDC</i>	On entry, <i>LDC</i> specifies the first dimension of <i>C</i> as declared in the calling (sub) program; <i>LDC</i> must be at least max(1, <i>N</i>); unchanged on exit.

STRMM, DTRMM, CTRMM, or ZTRMM Subroutine

Purpose

Performs matrix-matrix operations on triangular matrices.

Library

BLAS Library (**libblas.a**)

FORTRAN Syntax

```
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
```

SIDE, UPLO, TRANSA, DIAG
INTEGER *M, N, LDA, LDB*
COMPLEX*16 *ALPHA*
COMPLEX*16 *A(LDA,*)*, *B(LDB,*)*

Description

The **STRMM**, **DTRMM**, **CTRMM**, or **ZTRMM** subroutine performs one of the matrix-matrix operations:

$B := \alpha * \text{op}(A) * B$

OR

$B := \alpha * B * \text{op}(A)$

where α is a scalar, B is an M by 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 $\text{op}(A) = A^T$.

Parameters

SIDE On entry, *SIDE* specifies whether $\text{op}(A)$ multiplies B from the left or right as follows:

SIDE = 'L' or 'l'

$B := \alpha * \text{op}(A) * B$

SIDE = 'R' or 'r'

$B := \alpha * B * \text{op}(A)$

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'

$\text{op}(A) = A^T$

TRANSA = 'C' or 'c'

$\text{op}(A) = A^T$

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 α . When α is 0 then A is not referenced and B need not be set before entry; unchanged on exit.

<i>A</i>	An array of dimension (<i>LDA</i> , <i>k</i>), where <i>k</i> is <i>M</i> when <i>SIDE</i> = 'L' or 'l' and is <i>N</i> when <i>SIDE</i> = 'R' or 'r'; on entry with <i>UPLO</i> = 'U' or 'u', the leading <i>k</i> by <i>k</i> upper triangular part of the array <i>A</i> must contain the upper triangular matrix and the strictly lower triangular part of <i>A</i> is not referenced; on entry with <i>UPLO</i> = 'L' or 'l', the leading <i>k</i> by <i>k</i> lower triangular part of the array <i>A</i> must contain the lower triangular matrix and the strictly upper triangular part of <i>A</i> is not referenced. When <i>DIAG</i> = 'U' or 'u', the diagonal elements of <i>A</i> are not referenced either, but are assumed to be unity; unchanged on exit.
<i>LDA</i>	On entry, <i>LDA</i> specifies the first dimension of <i>A</i> as declared in the calling (sub) program. When <i>SIDE</i> = 'L' or 'l' then <i>LDA</i> must be at least max(1, <i>M</i>), when <i>SIDE</i> = 'R' or 'r' then <i>LDA</i> must be at least max(1, <i>N</i>); unchanged on exit.
<i>B</i>	An array of dimension (<i>LDB</i> , <i>N</i>); on entry, the leading <i>M</i> by <i>N</i> part of the array <i>B</i> must contain the matrix <i>B</i> , and on exit is overwritten by the transformed matrix.
<i>LDB</i>	On entry, <i>LDB</i> specifies the first dimension of <i>B</i> as declared in the calling (sub) program; <i>LDB</i> must be at least max(1, <i>M</i>); 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:

- $op(A) * X = alpha * B$
- $X * op(A) = alpha * B$

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 op(A) is either op(A) = A or op(A) = A'. The matrix X is overwritten on B.

Parameters

SIDE On entry, *SIDE* specifies whether op(A) appears on the left or right of X as follows:

SIDE = 'L' or 'l'
 $\text{op}(A) * X = \text{alpha} * B$

SIDE = 'R' or 'r'
 $X * \text{op}(A) = \text{alpha} * B$

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.

TRANS On entry, *TRANS* specifies the form of op(A) to be used in the matrix multiplication as follows:

TRANS = 'N' or 'n'
 $\text{op}(A) = A$

TRANS = 'T' or 't'
 $\text{op}(A) = A'$

TRANS = '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:

EACCES	Search permission is denied on a component of the path prefix.
EFAULT	The <i>Path</i> parameter points outside of the allocated address space of the process.
EIO	An I/O error occurred during the operation.
ELOOP	Too many symbolic links were encountered in translating the <i>Path</i> parameter.
ENAMETOOLONG	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.
ENOENT	A component of the path prefix does not exist.
ENOENT	A symbolic link was named, but the file to which it refers does not exist.
ENOENT	The path name is null.
ENOTDIR	A component of the path prefix is not a directory.
ESTALE	The root or current directory of the process is located in a virtual file system that is unmounted.

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 **odmerrno** variable is set to one of the following values:

ODMI_BAD_CLASSNAME	The specified object class name does not match the object class name in the file. Check path name and permissions.
ODMI_BAD_CLXNNAME	The specified collection name does not match the collection name in the file.
ODMI_BAD_CRIT	The specified search criteria is incorrectly formed. Make sure the criteria contains only valid descriptor names and the search values are correct. For information on qualifying criteria, see "Understanding ODM Object Searches" in <i>AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs</i> .
ODMI_BAD_LOCK	Cannot set a lock on the file. Check path name and permissions.
ODMI_BAD_TIMEOUT	The time-out value was not valid. It must be a positive integer.
ODMI_BAD_TOKEN	Cannot create or open the lock file. Check path name and permissions.
ODMI_CLASS_DNE	The specified object class does not exist. Check path name and permissions.
ODMI_CLASS_EXISTS	The specified object class already exists. An object class must not exist when it is created.
ODMI_CLASS_PERMS	The object class cannot be opened because of the file permissions.
ODMI_CLXNMAGICNO_ERR	The specified collection is not a valid object class collection.
ODMI_FORK	Cannot fork the child process. Make sure the child process is executable and try again.
ODMI_INTERNAL_ERR	An internal consistency problem occurred. Make sure the object class is valid or contact the person responsible for the system.
ODMI_INVALID_CLASS	The specified file is not an object class.
ODMI_INVALID_CLXN	Either the specified collection is not a valid object class collection or the collection does not contain consistent data.
ODMI_INVALID_PATH	The specified path does not exist on the file system. Make sure the path is accessible.
ODMI_LINK_NOT_FOUND	The object class that is accessed could not be opened. Make sure the linked object class is accessible.
ODMI_LOCK_BLOCKED	Cannot grant the lock. Another process already has the lock.
ODMI_LOCK_ENV	Cannot retrieve or set the lock environment variable. Remove some environment variables and try again.
ODMI_LOCK_ID	The lock identifier does not refer to a valid lock. The lock identifier must be the same as what was returned from the odm_lock subroutine.
ODMI_MAGICNO_ERR	The class symbol does not identify a valid object class.
ODMI_MALLOC_ERR	Cannot allocate sufficient storage. Try again later or contact the person responsible for the system.
ODMI_NO_OBJECT	The specified object identifier did not refer to a valid object.
ODMI_OPEN_ERR	Cannot open the object class. Check path name and permissions.
ODMI_OPEN_PIPE	Cannot open a pipe to a child process. Make sure the child process is executable and try again.
ODMI_PARAMS	The parameters passed to the subroutine were not correct. Make sure there are the correct number of parameters and that they are valid.
ODMI_READ_ONLY	The specified object class is opened as read-only and cannot be modified.
ODMI_READ_PIPE	Cannot read from the pipe of the child process. Make sure the child process is executable and try again.
ODMI_TOOMANYCLASSES	Too many object classes have been accessed. An application can only access less than 1024 object classes.
ODMI_UNLINKCLASS_ERR	Cannot remove the object class from the file system. Check path name and permissions.
ODMI_UNLINKCLXN_ERR	Cannot remove the object class collection from the file system. Check path name and permissions.
ODMI_UNLOCK	Cannot unlock the lock file. Make sure the lock file exists.

Related Information

List of ODM Commands and Subroutines in *AIX 5L Version 5.1 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.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

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.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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.

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.

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.

This edition applies to AIX 5L Version 5.1 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 Publications Department, Internal Zip 9561, 11400 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.

(c) Copyright AT&T, 1984, 1985, 1986, 1987, 1988, 1989. All rights reserved.

(c) Copyright KnowledgeSet Corporation, Mountainview, California, 1990.

Copyright (c) 1993, 1994 Hewlett-Packard Company

Copyright (c) 1993, 1994 International Business Machines Corp.

Copyright (c) 1993, 1994 Sun Microsystems, Inc.

Copyright (c) 1993, 1994 Novell, Inc.

All rights reserved. This product and related documentation are protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or related documentation may be reproduced in any form by any means without prior written authorization.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the United States Government is subject to the restrictions set forth in DFARS 252.227-7013 (c)(1)(ii) and FAR 52.227-19.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. HEWLETT-PACKARD COMPANY, INTERNATIONAL BUSINESS MACHINES CORP., SUN MICROSYSTEMS, INC., AND UNIX SYSTEMS LABORATORIES, INC., MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.

(c) Copyright Graphic Software Systems Incorporated, 1984, 1990. All rights reserved.

(c) Cornell University, 1989, 1990.

(c) Copyright Carnegie Mellon, 1988. All rights reserved.

(c) Copyright Stanford University, 1988. All rights reserved.

Permission to use, copy, modify, and distribute this program for any purpose and without fee is hereby granted, provided that this copyright and permission notice appear on all copies and supporting documentation, the name of Carnegie Mellon and Stanford University not be used in advertising or publicity pertaining to distribution of the program without specific prior permission, and notice be given in supporting documentation that copying and distribution is by permission of Carnegie Mellon and Stanford University. Carnegie Mellon and Stanford University make no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

This software and documentation is based in part on the Fourth Berkeley Software Distribution under license from The Regents of the University of California. We acknowledge the following institutions for their

role in its development: the Electrical Engineering and Computer Sciences Department at the Berkeley Campus.

The Rand MH Message Handling System was developed by the Rand Corporation and the University of California.

Portions of the code and documentation described in this book were derived from code and documentation developed under the auspices of the Regents of the University of California and have been acquired and modified under the provisions that the following copyright notice and permission notice appear:

Copyright Regents of the University of California, 1986, 1987, 1988, 1989. All rights reserved.

Redistribution and use in source and binary forms are permitted provided that this notice is preserved and that due credit is given to the University of California at Berkeley. The name of the University may not be used to endorse or promote products derived from this software without specific prior written permission. This software is provided "as is" without express or implied warranty.

Portions of the code and documentation described in this book were derived from code and documentation developed by Massachusetts Institute of Technology, Cambridge, Massachusetts, and Digital Equipment Corporation, Maynard, Massachusetts, and have been acquired and modified under the provision that the following copyright notice and permission notice appear:

(c) Copyright Digital Equipment Corporation, 1985, 1988, 1990, 1991. All rights reserved.

(c) Copyright 1985, 1986, 1987, 1988, 1989 Massachusetts Institute of Technology. All rights reserved.

Permission to use, copy, modify, and distribute this program and its documentation for any purpose and without fee is hereby granted, provided that this copyright, permission, and disclaimer notice appear on all copies and supporting documentation; the name of M.I.T. or Digital not be used in advertising or publicity pertaining to distribution of the program without specific prior permission. M.I.T. and Digital make no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

(c) Copyright Apollo Computer, Inc., 1987. All rights reserved.

(c) Copyright TITN, Inc., 1984, 1989. All rights reserved.

Index

Special Characters

`_lazySetErrorHandler` subroutine 525
`_safe_fetch` subroutine 79
`_showstring` subroutine 569
`_sync_cache_range` subroutine 281

Numerics

8-bit character capability 529

A

absolute values
 finding index of element with maximum value 613
access control information
 retrieving 245
access control subroutines
 `fstatacl` 245
 `revoke` 37
 `statacl` 245
`addch` subroutine 461
`addstr` subroutine 462
alarm signals
 beeping 468
 flashing 497
`alphasort` subroutine 80
alternate stack 146
asynchronous serial data line
 sending breaks on 318
`atoi` subroutine 269
`atol` subroutine 269
`attroff` subroutine 464
`attron` subroutine 466
`attrset` subroutine 467
authentication database
 opening and closing 119
authentication subroutines
 `endpwnb` 119
 `enduserdb` 124
 `setpwnb` 119
 `setuserdb` 124
 `tcb` 311

B

backspace character
 returning 495
`baudrate` subroutine 468
`beep` subroutine 468
Berkeley Compatibility Library
 subroutines
 `rand_r` 7
binary trees, manipulating 357
BLAS matrix-matrix operations 655
BLAS matrix-matrix subroutines 649, 650, 652, 654,
 657, 658, 660, 662

BLAS matrix-vector subroutines 617, 618, 620, 621,
 623, 624, 625, 626, 627, 629, 631, 632, 634, 636,
 638, 639, 640, 641, 642, 643, 644, 645, 646, 648
BLAS vector-vector functions 605, 606, 611, 612, 613,
 614
BLAS vector-vector subroutines 607, 608, 609, 610,
 612, 615, 616
`box` subroutine 469
buffers
 assigning to streams 100
bytes
 copying 274

C

carriage return 540
`CAXPY` subroutine 607
`cbox` subroutine 469
`cboxalt` subroutine 469
CBREAK mode 473
`cbreak` subroutine 473
`cbt` subroutine 211
`CCOPY` subroutine 609
`CDOTC` function 605
`CDOTU` function 606
`CGBMV` subroutine 618
`CGEMM` subroutine 649
`CGEMV` subroutine 617
`CGERC` subroutine 639
`CGERU` subroutine 639
change color definition 516
change color-pair definition 517
change terminal capabilities 496
character conversion
 wide characters
 lowercase to uppercase 334
 to double-precision number 399
 to long integer 403
 to multibyte 405, 412
 to tokens 401
 to unsigned long integer 406
 uppercase to lowercase 333
character data
 interpreting 82
 reading 82
character manipulation subroutines
 `vwsprintf` 384
character mapping 413
character transliteration 332
characters
 adding
 lines 520
 single characters 461, 519
 strings 462
 backspace 495
 clearing screen 475, 476
 controlling text scrolling 562, 563, 565
 deleting 489

- characters (*continued*)
 - dumping strings 569
 - echoing 492
 - erasing lines 479, 480, 490
 - erasing window 494
 - getting single characters 501
 - getting strings 506
 - handling input 529, 540
 - line-kill 524
 - placing at cursor location 514
 - reading formatted input 557
 - refreshing 592, 595
 - type ahead 598
 - typeahead 498
 - writing 384
 - writing formatted output 546
- charsetID
 - wide character 392
- CHBMV subroutine 621
- CHEMM subroutine 652
- CHEMV subroutine 620
- CHER subroutine 640
- CHER2 subroutine 642
- CHER2K subroutine 658
- CHERK subroutine 655
- CHPMV subroutine 623
- CHPR subroutine 641
- CHPR2 subroutine 643
- clear subroutine 475
- clearok subroutine 476
- close role database 121
- close SMIT ACL database 98
- closelog_r subroutine 305
- closelog subroutine 301
- clrtobot subroutine 479
- clrtoeol subroutine 480
- code sets
 - reading map files 102
- color, initialize 580
- color definition 516
- color intensity 481
- color manipulation 470
- color pair 544
- color-pair definition 517
- color support 511
- columns
 - determining number 568, 584
- compare wide character 443
- complex dot products
 - determining 605, 606
- control characters
 - specifying 599
- control input characters 510
- convert wide character 411
- converter subroutines
 - wcsrtombs 397
- copy a window region 482
- copy wide character 444
- cos subroutine 162
- cosh subroutine 163
- coshl subroutine 163
- cosl subroutine 162
- create subwindows 581
- cresetty subroutine 553
- CROTG subroutine 607
- CSCAL subroutine 612
- CSROT subroutine 608
- CSSCAL subroutine 612
- CSWAP subroutine 610
- CSYMM subroutine 650
- CSYR2K subroutine 657
- CSYRK subroutine 654
- CTBMV subroutine 629
- CTBSV subroutine 634
- CTPMV subroutine 631
- CTPSV subroutine 636
- CTRMM subroutine 660
- CTRMV subroutine 627
- CTRSM subroutine 662
- CTRSV subroutine 632
- cube roots
 - computing 211
- current process credentials
 - setting 111
- current process environment
 - setting 113
- current processes
 - group ID
 - setting 105
 - suspending 164
 - user information 373
- current screen
 - refreshing 550, 603
- current screens
 - refreshing 545
- curses
 - initializing 518
 - terminating 494
- curses character control subroutines
 - _showstring 569
 - addch 461
 - addstr 462
 - clear 475
 - clearok 476
 - clrtobot 479
 - clrtoeol 480
 - delch 489
 - deleteln 490
 - erase 494
 - getch 501
 - getstr 506
 - inch 514
 - insch 519
 - insertln 520
 - meta 529
 - mvaddch 461
 - mvaddstr 462
 - mvdelch 489
 - mvgetch 501
 - mvgetstr 506
 - mvinch 514
 - mvinsch 519

curses character control subroutines (*continued*)

mvscanw 557
mvwaddch 461
mvwaddstr 462
mvwdelch 489
mvwgetch 501
mvwgetstr 506
mvwinch 514
mvwinsch 519
mvwscanw 557
nodelay 540
scanw 557
scroll 562
scrollok 563
setscreg 565
unctrl 599
waddch 461
waddstr 462
wclear 475
wclrtoobot 479
wclrtoeol 480
wdelch 489
wdeleteln 490
werase 494
wgetch 501
wgetstr 506
winch 514
winsch 519
winsertln 520
wscanw 557
wsetscreg 565

curses cursor control subroutines

getyx 509
leaveok 526
move 530
mcur 531
wmove 530

curses data structure 560

curses options setting subroutines

idlok 513
intrflush 521
keypad 523
typeahead 598

curses portability subroutines

baudrate 468
erasechar 495
flushinp 498
killchar 524

curses subroutine

getbegyx 500
getmaxyx 505

curses subroutines

character locations
 echochar, wechochar, pechochar 493
endwin 494
initscr 518
switching input/output to different terminals 567

curses terminal manipulation subroutines

cbreak 473
cresetty 553
def_prog_mode 484

curses terminal manipulation subroutines (*continued*)

def_shell_mode 485
delay_output 488
echo 492
has_ic 512
has_il 513
longname 528
newterm 536
nl 540
nocbreak 473
noecho 492
nonl 540
noraw 549
putp 548
raw 549
reset_prog_mode 551
reset_shell_mode 552
resetterm 553
resetty 553
set_term 567
setupterm 568
tgetent 584
tgetflag 585
tgetnum 586
tgetstr 587
tgoto 588
tparm 596
tputs 597

curses video attributes subroutines

attroff 464
attron 466
attrset 467
beep 468
flash 497
standend 579
standout 579
vidattr 601
vidputs 601
wattroff 464
wattron 466
wattrset 467
wstandend 579
wstandout 579

curses window manipulation subroutines

box 469
delwin 491
doupdate 603
makenew 528
mvwin 532
newpad 533
newwin 537
overlay 543
overwrite 543
pnoutrefresh 545
prefresh 545
refresh 550
subwin 582
touchline 592
touchoverlap 594
touchwin 595
wnoutrefresh 602

curses window manipulation subroutines (*continued*)
wrefresh 550
cursor control
moving logical cursor 530
moving physical cursor 531
placing cursor 526
returning logical cursor coordinates 509
cursor coordinates 500
cursor visibility 483

D

D cache 281
DASUM subroutine 612
data
sorting with quicker-sort algorithms 1
data sorting subroutines
qsort 1
tdelete 357
tfind 357
tsearch 357
twalk 357
data transmissions
suspending 313
waiting for completion 312
data words
trace 348
databases
authentication
opening and closing 119
date
format conversions 262
date format conversions 271, 391
DAXPY subroutine 607
DCOPY subroutine 609
DDOT function 605
def_prog_mode subroutine 484
def_shell_mode subroutine 485
defect 220643 332
define character mapping 413
delay mode 510
delay_output subroutine 488
delch subroutine 489
deleteln subroutine 490
delwin subroutine 491
determine terminal color support 511
device driver
calling 288
device switch tables
checking entry status 297
DGBMV subroutine 618
DGEMM subroutine 649
DGEMV subroutine 617
DGER subroutine 638
directories
reading 14
removing 38
removing entries 369
renaming 34
scanning contents 80
sorting contents 80
directory subroutines
alphasort 80

directory subroutines (*continued*)
readlink 15
rmdir 38
scandir 80
symlink 278
unlink 369
disable terminal capabilities 496
discard lines in windows 499
disk quotas
manipulating 2
DNRM2 function 611
dot products
determining 605, 614
doupdate subroutine 603
drawbox subroutine 469
drawboxalt subroutine 469
DROT subroutine 608
DROTG subroutine 607
DROTM subroutine 615
DROTMG subroutine 616
DSBMV subroutine 625
DSCAL subroutine 612
DSPMV subroutine 626
DSPR subroutine 645
DSPR2 subroutine 648
DSWAP subroutine 610
DSYMM subroutine 650
DSYMV subroutine 624
DSYR subroutine 644
DSYR2 subroutine 646
DSYR2K subroutine 657
DSYRK subroutine 654
DTBMV subroutine 629
DTBSV subroutine 634
DTPMV subroutine 631
DTPSV subroutine 636
DTRMM subroutine 660
DTRMV subroutine 627
DTRSM subroutine 662
DTRSV subroutine 632
dump file, data structure 560
dump file, restore screen 561
DZASUM subroutine 612
DZNRM2 function 611

E

echo subroutine 492
echochar subroutine 493
echoing characters 492
endpwdb subroutine 119
endroledb subroutine 121
enduserdb subroutine 124
endwin subroutine 494
equations
solving systems 632, 634, 636
erase subroutine 494
erasechar subroutine 495
error codes 665
error codes, ODM 667
error handler, install 525
error handling
controlling system logs 301

- error handling (*continued*)
 - numbering error message string 259
- errorlogging_r subroutines 305
- errorlogging subroutines
 - closelog 301
 - openlog 301
 - setlogmask 301
 - syslog 301
- Euclidean lengths
 - determining 611
- examine state of alternate stack 146
- execution control
 - saving and restoring context 106
- execution control subroutines
 - longjmp 106
 - setjmp 106
- extended curses
 - initializing 518
- extended curses character control subroutines
 - _showstring 569
 - getch 501
 - inch 514
 - insch 519
 - meta 529
 - mvgetch 501
 - mvinch 514
 - mvinsch 519
 - mvscanw 557
 - mvwgetch 501
 - mvwinch 514
 - mvwinsch 519
 - mvwscanw 557
 - printw 546
 - scanw 557
 - scroll 562
 - scrollok 563
 - wgetch 501
 - winch 514
 - winsch 519
 - wscanw 557
- extended curses options setting subroutines
 - idlok 513
 - intrlflush 521
- extended curses portability subroutines
 - baudrate 468
 - erasechar 495
 - flushinp 498
 - killchar 524
- extended curses subroutines
 - initscr 518
- extended curses terminal manipulation subroutines
 - delay_output 488
 - has_ic 512
 - has_il 513
 - newterm 536
 - putp 548
 - resetterm 553
 - set_term 567
 - setupterm 568
 - tgentent 584
 - tgetflag 585
- extended curses terminal manipulation subroutines (*continued*)
 - tgetnum 586
 - tparm 596
- extended curses video attributes subroutines
 - attroff 464
 - attron 466
 - attrset 467
 - standend 579
 - standout 579
 - vidputs 601
 - wattroff 464
 - wattron 466
 - wattrset 467
 - wstandend 579
 - wstandout 579
- extended curses window manipulation subroutines
 - box 469
 - cbox 469
 - cboxalt 469
 - delwin 491
 - doupdate 602
 - drawbox 469
 - drawboxalt 469
 - fullbox 469
 - makenew 528
 - mvwin 532
 - newwin 537
 - overlay 543
 - overwrite 543
 - superbox 469
 - superbox1 469
 - touchline 592
 - touchoverlap 594
 - wnoutrefresh 602

F

- ffullstat subroutine 251
- file, input/output 558
- file access times
 - setting 375
- file creation masks
 - getting or setting values 364
- file descriptors
 - checking I/O status 87
- file modification times
 - setting 375
- file subroutines
 - ffullstat 251
 - fstat 251
 - fstatx 251
 - ftruncate 354
 - fullstat 251
 - lstat 251
 - remove 34
 - rename 34
 - stat 251
 - statx 251
 - tempnam 331
 - tmpfile 330

- file subroutines (*continued*)
 - tmpnam 331
 - truncate 354
 - umask 364
 - utime 375
 - utimes 375
- file system information 250
- file system subroutines
 - fstats 248
 - mount 380
 - quotactl 2
 - stats 248
 - sync 280
 - sysconf 282
 - umount 365
 - ustat 248
 - uvmount 365
 - vmount 380
- file systems
 - manipulating disk quotas 2
 - mounting 380
 - returning statistics 248
 - unmounting 365
 - updating 280
- files
 - changing length of regular 354
 - constructing names for temporary 331
 - creating symbolic links 278
 - creating temporary 330
 - deleting 34
 - providing status information 251
 - reading 9
 - removing 34
 - renaming 34
 - revoking access 37
 - writing to 449
- find wide character 442
- find wide character substring 399
- flash subroutine 497
- flow control
 - performing 313
- flushing
 - typeahead characters 498
- flushinp subroutine 498
- foreground process group IDs
 - getting 317
 - setting 321
- formatted input
 - converting 82
- fscanf subroutine 82
- fstat subroutine 251
- fstatacl subroutine 245
- fstats subroutine 248
- fstatvfs subroutine 250
- fstatx subroutine
 - described 251
- ftruncate subroutine 354
- fullbox subroutine 469
- fullstat subroutine 251

G

- get capabilities, terminfo 588
- get key name 522
- get terminals numeric value 590
- get terminals string capability 591
- get_wctype subroutine 413
- get_XTI variables 347
- getbegyx subroutine 500
- getch subroutine 501, 529, 540
- getmaxyx subroutine 505
- getstr subroutine 506
- getyx macro 509
- Givens plane rotations
 - constructing 607
- Givens transformations
 - applying 615
 - constructing 616
- gsignal subroutine 244
- gtty subroutine 273

H

- half-delay mode 510
- has_ic subroutine 512
- has_il subroutine 513
- Hermitian operations
 - performing rank 1 640, 641
 - performing rank 2 642, 643
 - performing rank 2k 658
 - performing rank k 655
- highlight mode 579
- hook words
 - trace 348
- hyperbolic functions
 - computing 163

I

- I cache 281
- I/O asynchronous subroutines
 - select 87
- I/O low-level subroutines 9, 449
 - readv 9
 - readx 9
 - writex 449
 - writex 449
- I/O stream subroutines
 - fscanf 82
 - scanf 82
 - setbuf 100
 - setbuffer 100
 - setlinebuf 100
 - setvbuf 100
 - sscanf 82
 - ungetc 368
 - ungetwc 368
 - wsscanf 82
- I/O terminal subroutines
 - gtty 273
 - isatty 360
 - stty 273

I/O terminal subroutines (*continued*)

- tcdrain 312
- tcflow 313
- tcflush 315
- tcgetattr 316
- tcgetpgrp 317
- tcsendbreak 318
- tcsetattr 319
- tcsetpgrp 321
- termdef 322
- ttylock 359
- ttylocked 359
- ttyname 360
- tty slot 361
- ttyunlock 359
- ttywait 359
- ICAMAX subroutine 613
- IDAMAX subroutine 613
- idlok subroutine 513
- idxpg4 255
- inch subroutine 514
- index subroutine 266
- initialize color 580
- initscr subroutine 518
- initstate subroutine 8
- input streams
 - pushing single character into 368
- insch subroutine 519
- insert-character capability 512
- insert/delete line option 513
- insert-line capability 513
- insertln subroutine 520
- interval timers
 - releasing 33
- intrflush subroutine 521
- ISAMAX subroutine 613
- isatty subroutine 360
- IZAMAX subroutine 613

J

- JFS
 - manipulating disk quotas 2

K

- kernel configurations
 - customizing 286
- kernel extension modules
 - loading 301
- kernel extensions
 - loading 291
- kernel object files
 - determining status 298
 - invoking 289
 - unloading 296
- kernel parameters
 - setting 299
- key name 522
- keypad
 - enabling 523
- keypad subroutine 523

- killchar subroutine 524

L

- label name, return 574
- lazy loading runtime system 525
- LC_ALL environment variable 108
- LC_COLLATE category 108
- LC_CTYPE category 108
- LC_MESSAGES category 108
- LC_MONETARY category 108
- LC_NUMERIC category 108
- LC_TIME category 108
- leaveok subroutine 526
- line-kill character 524
- lines
 - adding 520
 - determining number 568, 584
 - erasing 479, 480, 490
- links
 - creating symbolic 278
 - reading contents of symbolic 15
- locale subroutines
 - rpmatch 40
 - setlocale 107
- locales
 - changing or querying 107
 - response matching 40
- localization subroutines
 - strfmon 260
 - strftime 262
 - strptime 271
- locking functions
 - controlling tty 359
- logical cursor 509, 530
- long integers, converting
 - from character strings 269
 - from wide-character strings 403
- long numeric data 125
- longjmp subroutine 106
- longname subroutine 528
- lowercase characters
 - converting from uppercase 333
 - converting to uppercase 334
- lstat subroutine 251

M

- m_initscr subroutine 518
- makenew subroutine 528
- mapped files
 - attaching to process 126
- mapping, character 413
- matrices
 - performing matrix-matrix operations with
 - general matrices 649
 - Hermitian matrices 652
 - symmetric matrices 650
 - triangular matrices 660
 - performing matrix-vector operations with
 - general banded matrices 618
 - general matrices 617

- matrices (*continued*)
 - Hermitian band matrices 621
 - Hermitian matrices 620
 - packed Hermitian matrices 623
 - packed symmetric matrices 626
 - packed triangular matrices 631
 - symmetric band matrices 625
 - symmetric matrices 624
 - triangular band matrices 629
 - triangular matrices 627
- solving equations 662
- memory
 - freeing 448
- memory management
 - activating paging or swapping 275, 276
 - controlling shared memory operations 130
 - returning paging device status 277
 - returning shared memory segments 133
- memory management subroutines
 - shmat 126
 - shmctl 130
 - shmdt 132
 - shmget 133
 - swapoff 275
 - swapon 276
 - swapqry 277
- memory mapping
 - attaching segment or file to process 126
- message queues
 - checking I/O status 87
- meta subroutine 529
- minicurses
 - initializing 518
- minicurses subroutines
 - attrset 467
 - baudrate 468
 - erasechar 495
 - flushinp 498
 - getch 501
 - m_initscr 518
- monetary strings 260
- mount subroutine 380
- mounted file systems
 - returning statistics 248
- move subroutine 530
- multibyte characters
 - converting from wide 405, 412
- mvaddch subroutine 461
- mvaddstr subroutine 462
- mvcur subroutine 531
- mvdelch subroutine 489
- mvgetch subroutine 501
- mvgetstr subroutine 506
- mvinch subroutine 514
- mvinsch subroutine 519
- mvprintw subroutine 546
- mvscanw subroutine 557
- mvwaddch subroutine 461
- mvwaddstr subroutine 462
- mvwdelch subroutine 489
- mvwgetch subroutine 501

- mvwgetstr subroutine 506
- mvwin subroutine 532
- mvwinch subroutine 514
- mvwvinsch subroutine 519
- mvwprintw subroutine 546
- mvwscanw subroutine 557

N

- new-line character 540
- newpad subroutine 533
- newterm subroutine 536
- newwin subroutine 537
- nl subroutine 540
- no timeout mode 541
- nocbreak subroutine 473
- nodelay subroutine 540
- noecho subroutine 492
- nonl subroutine 540
- noraw subroutine 549
- nsleep subroutine 164
- numbers
 - generating
 - pseudo-random 5
 - random 5, 8
- numerical data
 - generating pseudo-random numbers 6
- numerical manipulation subroutines
 - atoi 269
 - atol 269
 - cbrt 211
 - cos 162
 - cosh 163
 - coshl 163
 - cosl 162
 - initstate 8
 - rand 5
 - random 8
 - rsqrt 77
 - setstate 8
 - sgetl 125
 - sin 162
 - sinh 163
 - sinhl 163
 - sinl 162
 - sputl 125
 - sqrt 211
 - sqrtl 211
 - srand 5
 - srandom 8
 - strtol 269
 - strtoul 269
 - tan 162
 - tanh 163
 - tanh1 163
 - tanl 162
 - watof 457
 - watoi 458
 - watol 458
 - wstrtod 457
 - wstrtol 458

O

- object file access subroutines
 - sgetl 125
 - sputl 125
- object file subroutines
 - unload 371
- object files
 - unloading 371
- Obtaining high-resolution elapsed time
 - read_real_time or time_base_to_time 17
- ODM error codes 667
- open role database 121
- open SMIT ACL database 98
- openlog_r subroutine 305
- openlog subroutine 301
- operating system
 - customizing configurations 286
 - identifying 367
- output
 - waiting for completion 312
- output, print 166
- overlay subroutine 543
- overwrite subroutine 543

P

- paging memory
 - activating 275, 276
 - returning information on devices 277
- parameter lists
 - handling variable-length 376
- parameter structures
 - copying into buffers 290
- path name
 - resolve 19
- path-name resolution 665
- pechochar subroutine 493
- performance data from remote kernels 79
- physical cursor 531
- plane rotations
 - applying 608
- pnoutrefresh subroutine 545
- prefresh subroutine 545
- print formatted output 166, 384
- printf subroutine 546
- printw subroutine 546
- process credentials
 - setting 111
- process environments
 - setting 113
- process group IDs
 - returning 317
 - setting 103, 117, 122, 321
 - supplementary IDs
 - setting 105
- process identification
 - current operating system name 367
- process initiation
 - restarting system 20
- process priorities
 - setting scheduled priorities 118

- process priorities (*continued*)
 - yielding to higher priorities 459
- process resource allocation
 - setting and getting user limits 362
- process signals
 - blocked signal sets
 - changing 158
 - returning 150
 - changing subroutine restart behavior 149
 - enhancement and management 153
 - handling system-defined exceptions 135
 - implementing software signal facility 244
 - manipulating signal sets 147
 - sending to executing program 4
- signal masks
 - replacing 158
 - saving or restoring 156
 - setting 151
- specifying action upon delivery 135
- stacks
 - defining alternate 157
 - saving or restoring context 156
- process subroutines (security and auditing)
 - setegid 103
 - seteuid 122
 - setgid 103
 - setgroups 105
 - setpcred 111
 - setpenv 113
 - setregid 103
 - setreuid 122
 - setrgid 103
 - setruid 122
 - setuid 122
 - system 310
 - usrinfo 373
- process user IDs
 - setting 122
- processes
 - handling user information 373
 - suspending 164, 385
- processes subroutines
 - gsignal 244
 - raise 4
 - reboot 20
 - semctl 90
 - semget 93
 - semop 96
 - setpgid 117
 - setpgrp 117
 - setpri 118
 - setsid 122
 - sigaddset 147
 - sigblock 151
 - sigdelset 147
 - sigemptyset 147
 - sigfillset 147
 - sighold 153
 - sigignore 153
 - siginterrupt 149
 - sigismember 147

- processes subroutines (*continued*)
 - siglongjmp 156
 - sigpause 158
 - sigpending 150
 - sigprocmask 151
 - sigreize 153
 - sigset 153
 - sigsetjmp 156
 - sigsetmask 151
 - sigstack 157
 - sigsuspend 158
 - ssignal 244
 - ulimit 362
 - uname 367
 - unamex 367
 - wait 385
 - wait3 385
 - waitpid 385
 - yield 459
- program mode 551
- pseudo-random numbers
 - generating 5
- pthread_kill subroutine 4
- push character to input queue 600
- putp subroutine 548

Q

- qsort subroutine 1
- queues
 - discarding data 315
- quotactl subroutine 2

R

- raise subroutine 4
- rand_r subroutine 6
- rand subroutine 5
- random numbers
 - generating 5, 8
- random subroutine 8
- rank 1 operations 638, 639
- raw mode 549
- raw subroutine 549
- re_comp subroutine 21
- re_exec subroutine 21
- re-initializest terminal structures 554
- read operations
 - from a file 9
- read protected data 79
- read_real_time Subroutine 17
- read subroutine 9
- readdir_r subroutine 14
- readlink subroutine 15
- readv subroutine
 - described 9
- readvx subroutine 9
- readx subroutine
 - described 9
- realpath subroutine 19
- reboot subroutine 20
- receive data unit 339

- reception of data
 - suspending 313
- reciprocals of square roots
 - computing 77
- refresh subroutine 550
- refreshing
 - characters 592, 595
 - current screen 545, 550, 603
 - standard screen 603
 - terminal 545, 550
 - windows 594, 603
- regcmp subroutine 22
- regcomp subroutine 25
- regerror subroutine 27
- regex subroutine 22
- regexec subroutine 29
- regfree subroutine 32
- regular expression subroutines
 - regcmp 22
 - regcomp 25
 - regerror 27
 - regex 22
 - regexec 29
 - regfree 32
- regular expressions
 - comparing 29
 - compiling 22, 25
 - error messages 27
 - freeing memory 32
 - matching 22
- regular files
 - changing length 354
- release indication
 - user data 335
- relinquish processor 87
- reltimerid subroutine 33
- remote hosts
 - rstat subroutine 79
- Remote Statistics Interface
 - subroutines
 - RSiChangeFeed 43
 - RSiChangeHotFeed 44
 - RSiClose 45
 - RSiCreateStatSet 47
 - RSiDelSetHot 48
 - RSiDelSetStat 50
 - RSiFirstCx 51
 - RSiFirstStat 52
 - RSiGetHotItem 54
 - RSiGetRawValue 56
 - RSiGetValue 57
 - RSiInit 58
 - RSiInstantiate 60
 - RSiMainLoop 63
 - RSiNextCx 64
 - RSiNextStat 65
 - RSiOpen 67
 - RSiPathAddSetStat 69
 - RSiPathGetCx 70
 - RSiStartFeed 71
 - RSiStartHotFeed 72

Remote Statistics Interface *(continued)*

- RSiStatGetPath 74
- RSiStopHotFeed 76
- remove subroutine 34
- rename subroutine 34
- replace lines in windows 499
- reserve a screen line 555
- reset_prog_mode subroutine 551
- reset_shell_mode subroutine 552
- resetterm subroutine 553
- resetty subroutine 553
- restore soft function key 577
- restore virtual screen 561
- retrieves information from terminfo 486
- return color intensity 481
- return file system information 250
- return label, soft label 574
- return window size 505
- returns color to color pair 544
- revoke subroutine 37
- rindex subroutine 266
- ripoffline subroutine 555
- rmdir subroutine 38
- rpmatch subroutine 40
- rsqrt subroutine 77
- rstat subroutine 79
- runtime tunable parameters
 - setting 299

S

- SASUM subroutine 612
- savetty subroutine 556
- SAXPY subroutine 607
- scandir subroutine 80
- scanf subroutine 82, 557
- scanw subroutine 557
- SCASUM subroutine 612
- sched_yield subroutine 87
- scheduling policy and priority
 - kernel thread 327
- SCNRM2 function 611
- SCOPY subroutine 609
- scr_dump subroutine 558
- scr_init subroutine 560
- scr_restore subroutine 561
- screen line 555
- screens
 - refreshing 545, 550, 603
- scroll subroutine 562
- scrollok subroutine 563
- SDOT function 605
- SDSDOT function 614
- select subroutine 87
- semaphore identifiers 93
- semaphore operations 90, 96
- semctl subroutine 90
- semget subroutine 93
- semop subroutine 96
- send data 341

- serial data lines
 - sending breaks on 318
- sessions
 - creating 122
- set blocking or non-blocking read 541
- set cursor visibility 483
- set_curterm subroutine 564
- set_term subroutine 567
- set terminal variables 564
- set wide character 445
- setaudithostdb or endaudithostdb subroutine 99
- setbuf subroutine 100
- setbuffer subroutine 100
- setcsmap subroutine 102
- setegid subroutine 103
- seteuid subroutine 122
- setgid subroutine 103
- setgroups subroutine 105
- setjmp subroutine 106
- setlinebuf subroutine 100
- setlocale subroutine 107
- setlogmask_r subroutine 305
- setlogmask subroutine 301
- setpcred subroutine 111
- setpenv subroutine 113
- setpgid subroutine 117
- setpgrp subroutine 117
- setpri subroutine 118
- setpwdb subroutine 119
- setregid subroutine 103
- setreuid subroutine 122
- setrgid subroutine 103
- setroledb subroutine 121
- setruid subroutine 122
- setscreg subroutine 565
- setsid subroutine 122
- setstate subroutine 8
- setsyx subroutine 566
- setuid subroutine 122
- setup soft labels 577
- setupterm subroutine 568
- setuserdb subroutine 124
- setvbuf subroutine 100
- SGBMV subroutine 618
- SGEMM subroutine 649
- SGEMV subroutine 617
- SGER subroutine 638
- sgetl subroutine 125
- shared memory segments
 - attaching to process 126
 - detaching 132
 - operations on 130
 - returning 133
- shell commands
 - running 310
- shell mode 485, 552
- shmat subroutine 126
- shmctl subroutine 130
- shmdt subroutine 132
- shmget subroutine 133

- short status requests
 - sending 233, 235
- sigaddset subroutine 147
- sigaltstack subroutine 146
- sigblock subroutine 151
- sigdelset subroutine 147
- sigemptyset subroutine 147
- sigfillset subroutine 147
- sighold subroutine 153
- sigignore subroutine 153
- siginterrupt subroutine 149
- sigismember subroutine 147
- siglongjmp subroutine 156
- signal masks
 - replacing 158
 - saving or restoring 156
 - setting 151
- signal stacks
 - defining alternate 157
 - saving or restoring context 156
- sigpause subroutine 158
- sigpending subroutine 150
- sigprocmask subroutine 151
- sigrelse subroutine 153
- sigset subroutine 153
- sigsetjmp subroutine 156
- sigsetmask subroutine 151
- sigstack subroutine 157
- sigsuspend subroutine 158
- sigwait subroutine 161
- sin subroutine 162
- single-byte conversion 411
- sinh subroutine 163
- sinhl subroutine 163
- sinl subroutine 162
- sleep subroutine 164
- slk_attroff subroutine 570
- slk_init subroutine 573
- slk_label subroutine 574
- slk_noutrefresh subroutine 575
- slk_refresh subroutine 576
- slk_restore subroutine 577
- slk_set subroutine 577
- slk_touch subroutine 578
- SMIT ACL database 98
- snprintf subroutine 166
- SNRM2 function 611
- soft function key, setup 577
- soft function key, update 578
- soft function key-label 573
- soft function key label, restore 577
- soft label, label name 574
- soft label, update 575, 576
- soft label subroutines 570
- sputl subroutine 125
- sqrt subroutine 211
- sqrtl subroutine 211
- square roots
 - computing 211
- srand subroutine 5
- random subroutine 8
- src_err_msg_r subroutine 213
- src_err_msg subroutine 212
- src error message
 - src error code 213
- SRC error messages
 - retrieving 212
- src request headers
 - return address 216
- SRC requests
 - getting subsystem reply information 214
 - sending replies 223
- SRC status text
 - returning title line 237
- SRC status text representations
 - getting 238, 239
- SRC subroutines
 - src_err_msg 212
 - srcrrqs 214
 - srcsbuf 217
 - srcsbuf_r 220
 - srcsrpy 223
 - srcsrqt 226
 - srcsrqt_r 229
 - srcstat 233
 - srcstat_r 235
 - srcstathdr 237
 - srcstattxt 238
 - srcstattxt_r 239
 - srcstop 239
 - srcstr 242
- srcrrqs_r subroutine 216
- srcrrqs subroutine 214
- srcsbuf_r subroutine 220
- srcsbuf subroutine 217
- srcsrpy subroutine 223
- srcsrqt_r subroutine 229
- srcsrqt subroutine 226
- srcstat_r subroutine 235
- srcstat subroutine 233
- srcstathdr subroutine 237
- srcstattxt_r subroutine 239
- srcstattxt subroutine 238
- srcstop subroutine 239
- srcstr subroutine 242
- SROT subroutine 608
- SROTG subroutine 607
- SROTM subroutine 615
- SROTMG subroutine 616
- SSBMV subroutine 625
- SSCAL subroutine 612
- sscanf subroutine 82
- ssignal subroutine 244
- SSPMV subroutine 626
- SSPR subroutine 645
- SSPR2 subroutine 648
- SSWAP subroutine 610
- SSYMM subroutine 650
- SSYMV subroutine 624
- SSYR subroutine 644
- SSYR2 subroutine 646
- SSYR2K subroutine 657

- SSYRK subroutine 654
- stack, alternate 146
- standard screen
 - clearing 475
 - refreshing 603
- standend subroutine 579
- standout subroutine 579
- start_color subroutine 580
- stat subroutine 251
- statacl subroutine 245
- stats subroutine 248
- statvfs subroutine 250
- statx subroutine 251
- STBMV subroutine 629
- STBSV subroutine 634
- store screen coordinates 500
- STPMV subroutine 631
- STPSV subroutine 636
- strcasecmp subroutine 258
- strcat subroutine 255
- strchr subroutine 266
- strcmp subroutine 257
- strcoll subroutine 258
- strcpy subroutine 256
- strcspn subroutine 266
- strdup subroutine 256
- streams
 - assigning buffers 100
- strerror subroutine 259
- strfmon subroutine 260
- strftime subroutine 262
- string conversion
 - to double-precision floating points 457
 - to integers 269, 458
 - to long integers 458
- string manipulation macros
 - varargs 376
- string manipulation subroutines
 - re_comp 21
 - re_exec 21
 - strncollen 268
 - wordexp 446
 - wordfree 448
 - wstring 454
- string operations
 - appending strings 255
 - comparing strings 257
 - copying strings 255
 - determining existence of strings 265
 - determining string location 265
 - determining string size 265
 - splitting strings into tokens 265
- string subroutines
 - index 266
 - rindex 266
 - strcasecmp 258
 - strcat 255
 - strchr 266
 - strcmp 257
 - strcoll 258
 - strcpy 256
- string subroutines *(continued)*
 - strcspn 266
 - strdup 256
 - strerror 259
 - strlen 266
 - strncasecmp 258
 - strncat 255
 - strncmp 258
 - strncpy 256
 - strpbrk 266
 - strrchr 266
 - strspn 266
 - strstr 266
 - strtok 266
 - strxfrm 255
- strings
 - compiling for pattern matching 21
 - performing operations on type wchar 454
 - returning number of collation values 268
- strlen subroutine 266
- STRMM subroutine 660
- STRMV subroutine 627
- strncasecmp subroutine 258
- strncat subroutine 255
- strncmp subroutine 258
- strncollen subroutine 268
- strncpy subroutine 256
- strpbrk subroutine 266
- strptime subroutine 271
- strrchr subroutine 266
- STRSM subroutine 662
- strspn subroutine 266
- strstr subroutine 266
- STRSV subroutine 632
- strtok subroutine 266
- strtol subroutine 269
- strtoul subroutine 269
- strxfrm subroutine 255
- stty subroutine 273
- subpad subroutine 581
- subroutines
 - remote statistics interface
 - RSiChangeFeed 43
 - RSiChangeHotFeed 44
 - RSiClose 45
 - RSiCreateStatSet 47
 - RSiDelSetHot 48
 - RSiDelSetStat 50
 - RSiFirstCx 51
 - RSiFirstStat 52
 - RSiGetHotItem 54
 - RSiGetRawValue 56
 - RSiGetValue 57
 - RSiInit 58
 - RSiInstantiate 60
 - RSiMainLoop 63
 - RSiNextCx 64
 - RSiNextStat 65
 - RSiOpen 67
 - RSiPathAddSetStat 69
 - RSiPathGetCx 70

- subroutines (*continued*)
 - RSiStartFeed 71
 - RSiStartHotFeed 72
 - RSiStatGetPath 74
 - RSiStopHotFeed 76
- restart behavior 149
- SPMI interface
 - SpmiAddSetHot 170
 - SpmiCreateHotSet 173
 - SpmiCreateStatSet 174
 - SpmiDdsAddCx 175
 - SpmiDdsDelCx 176
 - SpmiDdsInit 178
 - SpmiDelSetHot 179
 - SpmiDelSetStat 181
 - SpmiExit 182
 - SpmiFirstCx 183
 - SpmiFirstHot 184
 - SpmiFirstStat 185
 - SpmiFirstVals 186
 - SpmiFreeHotSet 187
 - SpmiFreeStatSet 188
 - SpmiGetCx 189
 - SpmiGetHotSet 191
 - SpmiGetStat 192
 - SpmiGetStatSet 193
 - SpmiGetValue 195
 - SpmiInit 196
 - SpmiInstantiate 197
 - SpmiNextCx 199
 - SpmiNextHot 200
 - SpmiNextHotItem 201
 - SpmiNextStat 203
 - SpmiNextVals 204
 - SpmiNextValue 205
 - SpmiPathAddSetStat 207
 - SpmiPathGetCx 209
 - SpmiStatGetPath 210
- subservers 217, 220
- substring, wide character 399
- subsystems
 - getting status 217, 220
 - returning status 233, 235
 - sending requests 226, 229
 - starting 242
 - stopping 239
- subwin subroutine 582
- subwindows 581
- superbox subroutine 469
- superbox1 subroutine 469
- supplementary process group IDs
 - setting 105
- swab subroutine 274
- swapoff subroutine 275
- swapon subroutine 276
- swapping memory
 - activating 275, 276
 - returning information on devices 277
- swapqpry subroutine 277
- symbolic links
 - creating 278
- symbolic links (*continued*)
 - reading contents 15
- symlink subroutine 278
- symmetric operations
 - performing rank 1 644, 645
 - performing rank 2 646, 648
 - performing rank 2k 657
 - performing rank k 654
- sync subroutine 280
- synchronize I cache with D cache 281
- SYS_CFGDD operation 288
- SYS_CFGKMOD operation 289
- SYS_GETPARMS operation 290
- SYS_KLOAD operation 291
- SYS_KULOAD operation 296
- SYS_QDVSW operation 297
- SYS_QUERYLOAD operation 298
- SYS_SETPARMS operation 299
- SYS_SINGLELOAD operation 301
- sysconf subroutine 282
- sysconfig operations
 - SYS_CFGDD 288
 - SYS_CFGKMOD 289
 - SYS_GETPARMS 290
 - SYS_KLOAD 291
 - SYS_KULOAD 296
 - SYS_QDVSW 297
 - SYS_QUERYLOAD 298
 - SYS_SETPARMS 299
 - SYS_SINGLELOAD 301
- sysconfig subroutine 286
- syslog_r subroutine 305
- syslog subroutine 301
- system limits
 - determining values 282
- System Performance Measurement Interface
 - subroutines
 - SpmiAddSetHot 170
 - SpmiCreateHotSet 173
 - SpmiCreateStatSet 174
 - SpmiDdsAddCx 175
 - SpmiDdsDelCx 176
 - SpmiDdsInit 178
 - SpmiDelSetHot 179
 - SpmiDelSetStat 181
 - SpmiExit 182
 - SpmiFirstCx 183
 - SpmiFirstHot 184
 - SpmiFirstStat 185
 - SpmiFirstVals 186
 - SpmiFreeHotSet 187
 - SpmiFreeStatSet 188
 - SpmiGetCx 189
 - SpmiGetHotSet 191
 - SpmiGetStat 192
 - SpmiGetStatSet 193
 - SpmiGetValue 195
 - SpmiInit 196
 - SpmiInstantiate 197
 - SpmiNextCx 199
 - SpmiNextHot 200

System Performance Measurement Interface
(continued)

- SpmiNextHotItem 201
- SpmiNextStat 203
- SpmiNextVals 204
- SpmiNextValue 205
- SpmiPathAddSetStat 207
- SpmiPathGetCx 209
- SpmiStatGetPath 210

system subroutine 310

T

- t_rcvreldata
 - subroutine 335
- t_rcvv subroutine 337
- t_rcvvudata subroutine 339
- t_sndreldata
 - subroutine 343
- t_sndv subroutine 341
- t_sndvudata
 - subroutine 345
- t_sysconf subroutine 347
- tables
 - sorting data 1
- tahn subroutine 163
- tan subroutine 162
- tanh1 subroutine 163
- tan1 subroutine 162
- TCB attributes
 - querying or setting 311
- tcb subroutine 311
- tcdrain subroutine 312
- tcflow subroutine 313
- tcflush subroutine 315
- tcgetattr subroutine 316
- tcgetpgrp subroutine 317
- tcsendbreak subroutine 318
- tcsetattr subroutine 319
- tcsetpgrp subroutine 321
- tdelete subroutine 357
- tempnam subroutine 331
- temporary files
 - constructing names 331
 - creating 330
- termcap identifiers
 - returning Boolean entry 585
 - returning numeric entry 586
 - returning string entry 587
- termdef subroutine 322
- terminal attributes
 - getting 316
 - setting 319
- terminal capabilities
 - applying parameters to 588, 596
 - insert-character capability 512
 - insert-line capability 513
- terminal capabilities, disable 496
- terminal color support 511
- terminal manipulation
 - determining number of lines and columns 568, 584
 - echoing characters 492

- terminal manipulation (continued)
 - outputting string with padding information 548, 597
 - switching input/output of curses subroutines 567
 - toggleing new-line and return translation 540
- terminal modes
 - CBREAK 473
 - program 551
 - raw 549
 - resetting 553
 - saving 484
 - shell 485, 552
- terminal names 360
- terminal numeric capability 590
- terminal speed 468
- terminal srting capability 591
- terminal states
 - getting 273, 316
 - setting 273, 319
- terminal structures 554
- terminal variables 564
- terminals
 - beeping 468
 - delaying output to 488
 - determining type 360
 - flashing 497
 - getting names 360
 - putting in video attribute mode 601
 - querying characteristics 322
 - refreshing 545, 550
 - setting up 536
 - verbose name 528
- terminfo database 588
- tfind subroutine 357
- tgetent subroutine 584
- tgetflag subroutine 585
- tgetnum subroutine 586
- tgetstr subroutine 587
- tgoto subroutine 588
- Thread-Safe C Library
 - subroutines
 - rand_r 7
 - readdir_r 14
- thread_self subroutine 327
- thread_setsched subroutine 327
- Threads Library
 - signal, sleep, and timer handling
 - raise subroutine 4
 - sithreadmask subroutine 159
 - sigwait subroutine 161
- tigetflag subroutine 588
- tigetnum subroutine 590
- tigetstr subroutine 591
- time_base_to_time Subroutine 17
- time format conversions 262, 271, 391
- time manipulation subroutines
 - nsleep 164
 - reltimerid 33
 - sleep 164
 - usleep 164
- time stamps
 - trace 348

- time subroutines
 - read_real_time 17
 - time_base_to_time 17
- timeout mode 541
- timezone subroutine 324
- tmpfile subroutine 330
- tmpnam subroutine 331
- touchline subroutine 592
- touchoverlap subroutine 594
- touchwin subroutine 595
- towctrans subroutine 332
- towlower subroutine 333
- towupper subroutine 334
- tparam subroutine 596
- tputs subroutine 597
- trace channels
 - halting data collection 350
 - recording trace event for 348
 - starting data collection 351
- trace data
 - halting collection 350
 - recording 348
 - starting collection 351
- trace events
 - recording 348, 349
- trace sessions
 - starting 352
 - stopping 353
- trace subroutines
 - trcgen 348
 - trcgent 348
 - trchook 349
 - trcoff 350
 - trcon 351
 - trcstart 352
 - trcstop 353
 - utrchook 349
- transmission of data
 - suspending 313
 - waiting for completion 312
- trcgen subroutine 348
- trcgent subroutine 348
- trchook subroutine 349
- trcoff subroutine 350
- trcon subroutine 351
- trcstart subroutine 352
- trcstop subroutine 353
- trigonometric functions
 - computing 162
 - computing hyperbolic 163
- truncate subroutine 354
- Trusted Computing Base attributes
 - querying or setting 311
- tsearch subroutine 357
- tty (teletypewriter)
 - flushing driver queue 521
- tty devices
 - determining 360
- tty locking functions
 - controlling 359

- tty modes
 - restoring state 553
 - saving state 556
- tty subroutines
 - setcsmap 102
- ttylock subroutine 359
- ttylocked subroutine 359
- ttynam subroutine 360
- ttyslot subroutine 361
- ttyunlock subroutine 359
- ttywait subroutine 359
- twalk subroutine 357
- type-ahead characters
 - flushing 498
- type ahead check 598
- typeahead subroutine 598

U

- ulimit subroutine 362
- umask subroutine 364
- umount subroutine 365
- uname subroutine 367
- unamex subroutine 367
- unctrl subroutine 599
- ungetc subroutine 368
- ungetch subroutine 600
- ungetwc subroutine 368
- unlink subroutine 369
- unload subroutine 371
- unlockpt subroutine 372
- unsigned long integers
 - converting wide-character strings to 406
- update soft labels 575, 576, 578
- uppercase characters
 - converting from lowercase 334
 - converting to lowercase 333
- user database
 - opening and closing 124
- user information
 - getting and setting 373
- usleep subroutine 164
- usrinfo subroutine 373
- ustat subroutine 248
- utime subroutine 375
- utimes subroutine 375
- utmp file
 - finding current user slot in 361
- utrchook subroutine 349
- uvmount subroutine 365

V

- varargs macros 376
- vectors
 - computing constant times vector plus vector 607
 - copying X to Y 609
 - interchanging X and Y 610
 - returning complex dot products 605, 606
 - returning dot products 605, 614
 - returning sum of absolute values 612
 - scaling by constants 612

VFS (Virtual File System)
 mounting 380
 unmounting 365
vfwprintf subroutine 379
vidattr subroutine 601
video attributes
 alarm signals
 beeping 468
 flashing 497
 highlight mode 579
 putting terminal in specified mode 601
 setting 467
 turning off 464
 turning on 466
vidputs subroutine 601
Virtual File System 380
virtual screen cursor coordinates 508
vmount subroutine 380
vsprintf subroutine 384
vwsprintf subroutine 384

W

waddch subroutine 461
waddstr subroutine 462
wait subroutine 385
wait3 subroutine 385
waitpid subroutine 385
watof subroutine 457
watoi subroutine 458
watol subroutine 458
wattroff subroutine 464
wattron subroutine 466
wattrset subroutine 467
wclear subroutine 475
wclrtoebot subroutine 479
wclrtoeol subroutine 480
wcscat subroutine 388
wcschr subroutine 388
wcscmp subroutine 388
wcscoll subroutine 390
wcscpy subroutine 388
wcscspn subroutine 388
wcsftime subroutine 391
wcsid subroutine 392
wcslen subroutine 393
wcsncat subroutine 394
wcsncmp subroutine 394
wcsncpy subroutine 394
wcpbrk subroutine 395
wcsrchr subroutine 396
wcsspncpy 398
wcstod 399
wcstok 401
wcstol 403
wcstoll 403
wcstombs 405
wcstoul 406
wcsvcs 408
wcsvwidth 409
wcsvxfrm 410
wctomb 412
wctype 413
wcvwidth 415
wide character substring 399
wide character to single-byte 411
wide characters
 comparing strings 390
 converting
 from date and time 391
 lowercase to uppercase 334
 to double-precision number 399
 to long integer 403
 to multibyte 405, 412
 to tokens 401
 to unsigned long integer 406
 uppercase to lowercase 333
 determining display width 409, 415
wcsvxfrm subroutine 410
wctob subroutine 411
wctomb subroutine 412
wctrans subroutine 413
wctype subroutine 413
wcvwidth subroutine 415
wdelch subroutine 489
wdeleteln subroutine 490
wechochar subroutine 493
werase subroutine 494
wgetch subroutine 501
wgetstr subroutine 506
wide character, memory 442, 443, 444, 445
wide character output 379
wide character subroutines
 get_wctype 413
 towlower 333
 toupper 334
 ungetc 368
 ungetwc 368
 wcscat 388
 wcschr 388
 wcscmp 388
 wcscoll 390
 wcscpy 388
 wcscspn 389
 wcsftime 391
 wcsid 392
 wcslen 393
 wcsncat 394
 wcsncmp 394
 wcsncpy 394
 wcpbrk 395
 wcsrchr 396
 wcsspncpy 398
 wcstod 399
 wcstok 401
 wcstol 403
 wcstoll 403
 wcstombs 405
 wcstoul 406
 wcsvcs 408
 wcsvwidth 409
 wcsvxfrm 410
 wctomb 412
 wctype 413
 wcvwidth 415
wide character substring 399
wide character to single-byte 411
wide characters
 comparing strings 390
 converting
 from date and time 391
 lowercase to uppercase 334
 to double-precision number 399
 to long integer 403
 to multibyte 405, 412
 to tokens 401
 to unsigned long integer 406
 uppercase to lowercase 333
 determining display width 409, 415

- wide characters (*continued*)
 - determining number in string 393
 - locating character sequences 408
 - locating single characters 396
 - obtaining handle for valid property names 413
 - operations on null-terminated strings 389, 394
 - pushing into input stream 368
 - returning charsetID 392
 - returning number in initial string segment 398
 - transforming strings to codes 410
- winch subroutine 514
- window, copy 482
- window coordinates 500
- window manipulation
 - creating structures
 - pad 533
 - subwindow 582
 - window 537
 - window buffer 528
 - drawing boxes 469
 - marking changed overlap 594
 - overwriting window 543
 - refreshing
 - characters 592, 595
 - current screen 545, 550, 603
 - standard screen 603
 - terminal 545, 550, 603
 - window 594, 602
- window size 505
- windows 499
 - clearing 475, 476
 - creating 537, 582
 - deleting 491
 - erasing 494
 - moving 532
 - refreshing 594, 602
 - scrolling 562, 563, 565
 - setting standout bit pattern 490
- winsch subroutine 519
- winsertln subroutine 520
- wmemchr subroutine 442
- wmemcmp subroutine 443
- wmemcpy subroutine 444
- wmemmove subroutine 444
- wmemset subroutine 445
- wmove subroutine 530
- wnoutrefresh subroutine 602
- word expansions
 - performing 446
- wordexp subroutine 446, 448
- wordfree subroutine 448
- wprintw subroutine 546
- wrefresh subroutine 550
- write contents of virtual screen 558
- write operations
 - writing to files 449
- write subroutine
 - described 449
- writev subroutine
 - described 449
- writevx subroutine 449

- writex subroutine
 - described 449
- wscanw subroutine 557
- wsetscrreg subroutine 565
- wsscanf subroutine 82
- wstandend subroutine 579
- wstandout subroutine 579
- wstring subroutines 454
- wstrtod subroutine 457
- wstrtol subroutine 458

X

- XTI variables 347

Y

- yield processor 87
- yield subroutine 459

Z

- ZAXPY subroutine 607
- ZCOPY subroutine 609
- ZDOTC function 605
- ZDOTU function 606
- ZDROT subroutine 608
- ZDSCAL subroutine 612
- ZGBMV subroutine 618
- ZGEMM subroutine 649
- ZGEMV subroutine 617
- ZGERC subroutine 639
- ZGERU subroutine 639
- ZHBMV subroutine 621
- ZHEMM subroutine 652
- ZHEMV subroutine 620
- ZHER subroutine 640
- ZHER2 subroutine 642
- ZHER2K subroutine 658
- ZHERK subroutine 655
- ZHPMV subroutine 623
- ZHPR subroutine 641
- ZHPR2 subroutine 643
- ZROTG subroutine 607
- ZSCAL subroutine 612
- ZSWAP subroutine 610
- ZSYMM subroutine 650
- ZSYR2K subroutine 657
- ZSYRK subroutine 654
- ZTBMV subroutine 629
- ZTBSV subroutine 634
- ZTPMV subroutine 631
- ZTPSV subroutine 636
- ZTRMM subroutine 660
- ZTRMV subroutine 627
- ZTRSM subroutine 662
- ZTRSV subroutine 632

Readers' Comments — We'd Like to Hear from You

AIX 5L Version 5.1

Technical Reference: Base Operating System and Extensions , Volume 2

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

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.

Readers' Comments — We'd Like to Hear from You



Cut or Fold
Along Line

Fold and Tape

Please do not staple

Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Corporation
Publications Department
Internal Zip 9561
11400 Burnet Road
Austin, TX
78758-3493

Fold and Tape

Please do not staple

Fold and Tape

Cut or Fold
Along Line



Printed in U.S.A